Programmation d'ADOBE® ACTIONSCRIPT® 3.0

Adobe®

Programmation avec Adobe® ActionScript® 3.0 pour Adobe® Flash®

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

Sommaire

Chapitre 1 : A propos de ce manuel

Ce manuel peut servir de base au développement d'applications avec Adobe® ActionScript® 3.0. Il est préférable de maîtriser les concepts de programmation généraux (types de données, variables, boucles et fonctions) afin de mieux comprendre les idées et les techniques décrites. Il est également conseillé de comprendre les concepts de la programmation orientée objets (classes et héritage). La connaissance d'ActionScript 1.0 ou ActionScript 2.0 est utile, mais pas nécessaire.

Utilisation de ce manuel

Les chapitres de ce manuel sont structurés en groupes logiques (indiqués ci-dessous) afin de faciliter la recherche de sections de documentation ActionScript connexes :

Chapitres	Description
Chapitres 2 à 5 : présentation de la programmation avec ActionScript	Décrit les concepts ActionScript 3.0 de base, y compris la syntaxe, les instructions et les opérateurs du langage, ainsi que la programmation ActionScript orientée objet
Chapitres 6 à 11 : classes et types de données ActionScript 3.0 de base	Décrit les types de données de niveau supérieur dans ActionScript 3.0.
Chapitres 12 à 32 : Flash Player et les interfaces de programmation d'Adobe AIR	Décrit les fonctions importantes implémentées dans des packages et des classes propres à Adobe Flash Player, notamment la gestion d'événement, l'utilisation d'objets d'affichage et de la liste d'affichage, la mise en réseau et les communications, le fichier d'entrée-sortie, l'interface externe, le modèle de sécurité de l'application, etc.

Ce manuel contient également de nombreux fichiers d'exemple qui décrivent des concepts de programmation d'applications pour des classes importantes ou couramment utilisées. Les fichiers d'exemple sont conçus pour une utilisation et un chargement plus facile avec Adobe® Flash® CS4 Professional. Ils peuvent inclure des fichiers enveloppe. Néanmoins, l'exemple de code de base est du code ActionScript 3.0 pur, que vous pouvez utiliser dans l'environnement de développement de votre choix.

ActionScript 3.0 peut être écrit et compilé de plusieurs façons différentes :

- En utilisant l'environnement de développement d'Adobe Flex Builder 3
- En utilisant un éditeur de texte et un compilateur de ligne de commande (celui fourni avec Flex Builder 3, par exemple)
- En utilisant l'outil de programmation Adobe® Flash® CS4 Professional

Pour plus d'informations sur les environnements de développement ActionScript, consultez « Introduction à ActionScript 3.0 » à la page 4.

Pour comprendre les exemples de code fournis dans ce manuel, vous n'avez pas besoin de savoir utiliser les environnements de développement intégrés pour ActionScript (Flex Builder ou l'outil de programmation de Flash, par exemple). En revanche, vous pouvez consulter la documentation relative à ces outils pour savoir comment les utiliser pour écrire et compiler du code ActionScript 3.0. Pour plus d'informations, consultez la section « Accès à la documentation ActionScript » à la page 2.

Accès à la documentation ActionScript

Etant donné que ce manuel traite principalement de la description d'ActionScript 3.0 qui est un langage de programmation orienté objet riche et puissant, il ne décrit pas de façon détaillée le processus de développement des applications ni le flux de travail au sein d'une architecture serveur ou d'un outil particulier. Par conséquent, outre la *Programmation avec ActionScript 3.0*, vous pouvez consulter d'autres sources de documentation lorsque vous concevez, développez, testez et déployez des applications ActionScript 3.0.

Documentation ActionScript 3.0

Ce manuel décrit les concepts qui sous-tendent le langage de programmation ActionScript 3.0 et fournit des détails et des exemples d'implémentation illustrant les fonctions importantes du langage. Néanmoins, il ne constitue pas un guide de référence du langage complet. A cet effet, consultez le Guide de référence du langage et des composants ActionScript 3.0, qui décrit chaque classe, chaque méthode, chaque propriété et chaque événement du langage. Le Guide de référence du langage et des composants ActionScript 3.0 fournit des informations de référence détaillées sur le langage de base, les composants de l'outil de programmation Flash (dans les packages fl) et les interfaces de programmation Flash Player et Adobe AIR (dans les packages flash).

Documentation Flash

Si vous utilisez l'outil de programmation, vous, vous pouvez consulter les manuels suivants :

Manuel	**Description**
Utilisation de Flash	Décrit comment développer vos applications Web dynamiques dans l'outil de programmation Flash
Programmation avec ActionScript 3.0	Décrit l'utilisation spécifique du langage ActionScript 3.0 et de l'interface de programmation de base de Flash Player et d'Adobe AIR
Guide de référence du langage et des composants ActionScript 3.0	Fournit des exemples de syntaxe, d'utilisation et de code pour les composants de l'outil de programmation Flash et l'interface de programmation ActionScript 3.0
Utilisation des composants ActionScript 3.0	Explique de façon détaillée comment utiliser des composants pour développer des applications créées par Flash.
Développement d'applications AIR avec Adobe Flash CS4 Professional	Explique comment développer et déployer des applications Adobe AIR par le biais d'ActionScript 3.0 et de l'API d'Adobe AIR dans Flash.
Formation à ActionScript 2.0 dans Adobe Flash	Passe en revue les principes généraux de la syntaxe ActionScript 2.0 et explique comment utiliser ce langage pour intervenir sur différents types d'objet
Guide de référence du langage ActionScript 2.0	Fournit des exemples de syntaxe, d'utilisation et de code pour les composants de l'outil de programmation Flash et l'interface de programmation ActionScript 2.0
Utilisation des composants ActionScript 2.0	Explique de façon détaillée comment utiliser les composants ActionScript 2.0 pour développer des applications créées par Flash.
Guide de référence du langage et des composants ActionScript 2.0	Décrit chaque composant disponible dans l'architecture des composants Adobe, version 2, avec son API
Extension de Flash	Décrit les objets, les méthodes et les propriétés disponibles dans l'API de JavaScript
Prise en main de Flash Lite 2.x	Explique comment utiliser Adobe® Flash® Lite™ 2.x pour développer des applications et fournit des exemples de code, d'utilisation et de syntaxe pour les fonctions ActionScript disponibles avec Flash Lite 2.x
Développement d'applications Flash Lite 2.x	Explique comment développer des applications Flash Lite 2.x

Manuel	Description
Présentation du code ActionScript pour Flash Lite 2.x	Explique comment développer des applications avec Flash Lite 2.x et décrit toutes les fonctions ActionScript disponibles pour les développeurs de Flash Lite 2.x
Guide de référence du langage ActionScript Flash Lite 2.x	Fournit des exemples de code, d'utilisation et de syntaxe pour l'API d'ActionScript 2.0 disponible dans Flash Lite 2.x
Prise en main de Flash Lite 1.x	Présente Flash Lite 1.x et décrit comment tester votre contenu à l'aide de l'émulateur CS4 d'Adobe® Device Central
Développement d'applications Flash Lite 1.x	Décrit comment développer des applications pour périphériques mobiles à l'aide de Flash Lite 1.x
Formation à ActionScript Flash Lite 1.x	Explique comment utiliser ActionScript dans des applications Flash Lite 1.x et décrit toutes les fonctions ActionScript disponibles avec Flash Lite 1.x
Référence du langage ActionScript Flash Lite 1.x	Fournit la syntaxe et l'utilisation d'éléments ActionScript disponibles avec Flash Lite 1.x

Ressources ActionScript destinées à la formation

Outre le contenu de ces manuels, Adobe fournit des articles, des idées de conception et des exemples mis à jour régulièrement dans le Pôle de développement et le Pôle de création Adobe.

Pôle de développement Adobe

Le Pôle de développement Adobe vous permet d'accéder aux informations les plus récentes sur ActionScript, aux articles sur le développement d'applications réelles et aux informations sur les problèmes importants. Accédez au Pôle de développement à l'adresse www.adobe.com/devnet/.

Pôle de création Adobe

Consultez les dernières nouveautés en matière de design numérique et d'animations. Parcourez les œuvres d'artistes renommés, découvrez les nouvelles tendances de design et renforcez vos connaissances à l'aide de didacticiels, de flux de travail clés et de techniques avancées. Connectez-vous deux fois par mois pour accéder aux didacticiels et aux articles les plus récents, ainsi qu'aux galeries qui seront votre source d'inspiration. Accédez au Pôle de création à l'adresse www.adobe.com/designcenter/.

Chapitre 2 : Introduction à ActionScript 3.0

Ce chapitre présente Adobe® ActionScript® 3.0, la version la plus récente et la plus révolutionnaire d'ActionScript.

A propos d'ActionScript

ActionScript est le langage de programmation des environnements d'exécution d'Adobe® Flash® Player et Adobe® AIR™. Il assure l'interactivité, le traitement des données et bien d'autres fonctions pour le contenu Flash, Flex et AIR et les applications associées.

ActionScript est exécuté par la machine virtuelle ActionScript, un composant de Flash Player et AIR. Le code ActionScript est généralement compilé en *pseudo-code binaire* (sorte de langage de programmation écrit et compris par les ordinateurs) par un compilateur, tel celui intégré à Adobe® Flash® CS4 Professional ou Adobe® Flex™ Builder™, ou celui fourni dans le kit de développement d'Adobe® Flex™. Le pseudo-code binaire est intégré aux fichiers SWF, qui sont exécutés par Flash Player et AIR.

ActionScript 3.0 constitue un modèle de programmation solide, bien connu des développeurs dotés des connaissances élémentaires sur la programmation orientée objets. Parmi les principales fonctionnalités d'ActionScript 3.0 qui ont été améliorées par rapport aux versions antérieures d'ActionScript figurent :

- Une nouvelle machine virtuelle ActionScript, appelée AVM2, qui exploite un nouveau jeu d'instructions de pseudo-code binaire et améliore grandement les performances.
- Un code de compilateur plus moderne qui effectue des optimisations de plus bas niveau que les versions antérieures du compilateur.
- Une interface de programmation (API) étendue et améliorée, avec contrôle de bas niveau des objets et un véritable modèle orienté objet.
- Une API XML reposant sur la spécification ECMAScript pour XML (E4X) (ECMA-357 niveau 2). E4X est une extension de langage d'ECMAScript qui ajoute XML comme type de données natif.
- Un modèle d'événements fondé sur la spécification d'événements du modèle d'objet de document (DOM, Document Object Model) niveau 3.

Avantages d'ActionScript 3.0

Les possibilités d'ActionScript 3.0 dépassent largement les fonctions de programmation des versions précédentes. Cette version est conçue pour faciliter la création d'applications très complexes impliquant d'importants jeux de données et des bases de code orientées objet et réutilisables. Si ActionScript 3.0 n'est pas indispensable à l'exécution de contenu dans Adobe Flash Player, il ouvre néanmoins la voie à des améliorations de performance uniquement disponibles dans AVM2, la nouvelle machine virtuelle. Le code d'ActionScript 3.0 peut s'exécuter jusqu'à dix fois plus vite que le code des versions antérieures d'ActionScript.

L'ancienne version de la machine virtuelle, AVM1, exécute le code ActionScript 1.0 et ActionScript 2.0. Elle est prise en charge par Flash Player 9 et 10 pour assurer la compatibilité ascendante avec le contenu existant. Pour plus d'informations, consultez la section « Compatibilité avec les versions précédentes » à la page 8.

Nouveautés d'ActionScript 3.0

Bien que de nombreuses classes et fonctions d'ActionScript 3.0 soient bien connues des programmeurs ActionScript, son architecture et sa conceptualisation diffèrent des versions précédentes. Parmi les améliorations d'ActionScript 3.0, on compte de nouvelles fonctions du langage de base et une API Flash Player avancée, qui accroît le contrôle des objets de bas niveau.

Remarque : *les applications Adobe® AIR™ peuvent également utiliser les API de Flash Player.*

Fonctions du langage de base

Le langage de base définit les éléments de construction fondamentaux du langage de programmation, par exemple les arguments, expressions, conditions, boucles et types. ActionScript 3.0 contient de nombreuses fonctions qui accélèrent le processus de développement.

Exceptions d'exécution

ActionScript 3.0 peut signaler davantage de conditions d'erreur que les versions précédentes. Utilisées pour les conditions d'erreur courantes, les exceptions d'exécution améliorent la procédure de débogage et vous permettent de développer des applications susceptibles de gérer les erreurs de manière fiable. Les erreurs d'exécution peuvent fournir des traces de pile qui identifient le fichier source et le numéro de ligne, pour un repérage plus rapide des erreurs.

Types d'exécution

Dans ActionScript 2.0, les annotations de type visaient avant tout à aider le développeur ; lors de l'exécution, toutes les valeurs étaient typées dynamiquement. Dans ActionScript 3.0, les informations de type sont préservées lors de l'exécution et utilisées à plusieurs fins. Flash Player et Adobe AIR vérifient les types lors de l'exécution, optimisant ainsi l'intégrité des types du système. Les informations de type servent également à représenter les variables dans les représentations machine natives, ce qui accroît les performances et réduit l'utilisation de la mémoire.

Classes scellées

ActionScript 3.0 introduit le concept de classe scellée. Une telle classe possède uniquement un jeu fixe de propriétés et de méthodes, définies lors de la compilation. Il est impossible d'en ajouter d'autres. Ainsi, la vérification effectuée au moment de la compilation est plus stricte et garantit une plus grande robustesse des programmes. L'utilisation de la mémoire est également optimisée puisqu'une table de hachage interne n'est pas requise pour chaque occurrence d'objet. Les classes dynamiques sont également disponibles par le biais du mot-clé `dynamic`. Bien que scellées par défaut, toutes les classes d'ActionScript 3.0 peuvent être déclarées dynamiques grâce au mot-clé `dynamic`.

Fermetures de méthodes

ActionScript 3.0 permet l'utilisation d'une fermeture de méthode qui se rappelle automatiquement l'occurrence de son objet d'origine. Cette fonction s'avère utile dans le traitement des événements. Dans ActionScript 2.0, les fermetures de méthode ne gardaient pas la trace de l'occurrence d'objet à partir de laquelle il était extrait, ce qui provoquait un comportement inattendu lors de l'appel de la fermeture de méthode. La classe mx.utils.Delegate était une solution prisée, qui n'est désormais plus nécessaire.

ECMAScript pour XML (E4X)

ActionScript 3.0 intègre ECMAScript pour XML (E4X), récemment normalisé sous le nom ECMA-357. E4X offre un jeu d'éléments de langage naturels et courants qui permettent de manipuler XML. Contrairement aux API classiques d'analyse XML, XML et E4X fonctionnent comme un type de données natif du langage. E4X simplifie le développement d'applications exploitant XML grâce à une réduction drastique du volume de code requis. Pour plus d'informations sur l'intégration d'E4X dans ActionScript 3.0, consultez le chapitre « Utilisation de XML » à la page 232.

Pour visualiser la spécification E4X d'ECMA, consultez le site www.ecma-international.org.

Expressions régulières

ActionScript 3.0 inclut une prise en charge native des expressions régulières afin d'accélérer la recherche et la manipulation des chaînes. Dans ActionScript 3.0, cette prise en charge suit la version 3 de la spécification de langage ECMAScript (ECMA-262).

Espaces de noms

Les espaces de noms sont semblables aux spécificateurs d'accès classiques qui assurent le contrôle de visibilité des déclarations (`public`, `private`, `protected`). Ils fonctionnent comme des spécificateurs d'accès personnalisés, qui portent le nom de votre choix. Les espaces de noms sont dotés d'un identifiant de ressource universel (URI, Universal Resource Identifier) afin d'éviter les collisions. Ils servent également à représenter les espaces de noms XML en cas d'utilisation d'E4X.

Nouveaux types de primitives

ActionScript 2.0 utilise un seul type numérique, Number, un nombre en virgule flottante à deux décimales. ActionScript 3.0 comprend les types int et uint. Le type int est un entier signé 32 bits qui permet au code ActionScript de profiter de la rapidité de traitement mathématique de l'unité centrale. Il s'avère pratique pour les compteurs de boucles et les variables utilisant des entiers. Le type uint est un type d'entier non signé 32 bits, utile pour les valeurs de couleurs RVB, les compteurs d'octets, etc.

Fonctions API de Flash Player

Les API Flash Player d'ActionScript 3.0 contiennent un grand nombre de classes qui vous permettent de contrôler les objets de bas niveau. L'architecture du langage est conçue pour être plus intuitive que les versions antérieures. Ces nouvelles classes étant trop nombreuses pour autoriser une présentation détaillée à ce stade, les sections ci-après mettent en avant quelques changements significatifs.

Remarque : *les applications Adobe® AIR™ peuvent également utiliser les API de Flash Player.*

Modèle d'événements DOM3

Le modèle d'événements Document Object Model de niveau 3 (DOM3) offre une méthode standard de génération et de traitement des messages d'événement. Il permet aux objets composant les applications d'interagir et de communiquer tout en conservant leur état et en réagissant aux changements. Etabli à partir des spécifications d'événements DOM niveau 3 du World Wide Web Consortium, ce modèle fournit un mécanisme plus clair et plus efficace que les systèmes d'événement disponibles dans les versions antérieures d'ActionScript.

Les événements et événements d'erreur se trouvent dans le package flash.events. La structure des composants Flash utilisant le même modèle d'événements que l'API de Flash Player, le système d'événements est unifié sur l'ensemble de la plate-forme Flash.

API de liste d'affichage

L'API d'accès à la liste d'affichage Flash Player et Adobe AIR, c'est-à-dire l'arborescence contenant tous les éléments visuels d'une application Flash, est constituée de classes permettant de manipuler les primitives visuelles dans Flash.

La nouvelle classe Sprite est un élément de construction léger, semblable à la classe MovieClip mais plus adaptée à la classe de base des composants d'interface. La nouvelle classe Shape représente des formes vectorielles brutes. Il est possible d'instancier ces classes naturellement à l'aide de l'opérateur `new`, mais aussi de les redéfinir dynamiquement comme parent à tout moment.

Grâce à la gestion de profondeur désormais automatique et intégrée à Flash Player et Adobe AIR, le rendu d'affectation de numéros de profondeur n'est plus nécessaire. De nouvelles méthodes permettent de spécifier et de gérer l'ordre z des objets.

Gestion des données et contenus dynamiques

ActionScript 3.0 comprend des mécanismes de chargement et de gestion des actifs et des données au sein de l'application qui se caractérisent par leur intuitivité et leur cohérence dans l'ensemble de l'API. La nouvelle classe Loader propose un unique mécanisme de chargement des fichiers SWF et des actifs d'image, et permet d'accéder à des informations détaillées sur le contenu chargé. La classe URLLoader offre un mécanisme distinct de chargement du texte et des données binaires dans les applications orientées données. La classe Socket permet la lecture et l'écriture des données binaires dans les sockets de serveur, quel que soit le format.

Accès aux données de bas niveau

De nombreuses API permettent d'accéder à des données de bas niveau jusqu'ici indisponibles dans ActionScript. Pour le téléchargement de données, la classe URLStream, implémentée par URLLoader, donne accès aux données sous forme binaire brute pendant le téléchargement. Avec la classe ByteArray, vous pouvez optimiser la lecture, l'écriture et la manipulation des données binaires. La nouvelle API Sound assure le contrôle précis du son par le biais des classes SoundChannel et SoundMixer. De nouvelles API liées à la sécurité fournissent des informations sur les droits de sécurité d'un fichier SWF ou du contenu chargé, pour une gestion plus efficace des erreurs de sécurité.

Utilisation de texte

ActionScript 3.0 contient un package flash.text destiné à l'ensemble des API relatives au texte. La classe TextLineMetrics propose des mesures détaillées relatives à une ligne de texte au sein d'un champ de texte. Elle remplace la méthode `TextFormat.getTextExtent()` intégrée à ActionScript 2.0. La classe TextField contient diverses méthodes de bas niveau intéressantes, qui peuvent fournir des informations déterminées sur une ligne de texte ou un caractère unique dans un champ de texte. Parmi ces méthodes figurent : `getCharBoundaries()`, qui renvoie un rectangle représentant le cadre de sélection d'un caractère,`getCharIndexAtPoint()`, qui renvoie l'index d'un caractère à un point donné et `getFirstCharInParagraph()`, qui renvoie l'index du premier caractère d'un paragraphe. Les méthodes de niveau de ligne incluent `getLineLength()`, qui renvoie le nombre de caractères d'une ligne de texte donnée, et `getLineText()`, qui renvoie le texte de la ligne spécifiée. La nouvelle classe Font permet de gérer les polices intégrées à des fichiers SWF.

Compatibilité avec les versions précédentes

Comme toujours, Flash Player garantit une compatibilité totale avec les versions précédentes et les contenus publiés antérieurement. Tout contenu qui s'exécutait dans les versions antérieures de Flash Player s'exécute dans Flash Player 9 et ultérieur. L'introduction d'ActionScript 3.0 dans Flash Player 9 présente toutefois certains problèmes de compatibilité entre le contenu créé dans des versions antérieures et tout nouveau contenu qui s'exécute dans Flash Player 9 ou ultérieur, Voici quelques problèmes de compatibilité :

- Un même fichier SWF ne peut combiner du code ActionScript 1.0 ou 2.0 et du code ActionScript 3.0.
- Le code ActionScript 3.0 peut charger un fichier SWF écrit en ActionScript 1.0 ou 2.0, mais ne peut accéder aux variables et fonctions de ce fichier.
- Les fichiers SWF écrits en ActionScript 1.0 ou 2.0 ne peuvent pas charger les fichiers SWF écrits en ActionScript 3.0. Cela signifie que les fichiers SWF créés dans Flash 8, Flex Builder 1.5 ou une version antérieure ne peuvent pas charger les fichiers SWF écrits en ActionScript 3.0.

  Il existe une seule exception à cette règle : un fichier SWF écrit en ActionScript 2.0 peut être remplacé par un fichier SWF écrit en ActionScript 3.0, sous réserve que le fichier écrit en ActionScript 2.0 n'ait effectué aucun chargement à quelque niveau que ce soit. Pour ce faire, le fichier SWF écrit en ActionScript 2.0 doit appeler `loadMovieNum()` et transmettre la valeur 0 au paramètre `level`.
- En règle générale, les fichiers SWF écrits en ActionScript 1.0 ou 2.0 doivent faire l'objet d'une migration pour fonctionner conjointement avec des fichiers SWF écrits en ActionScript 3.0. Supposons, par exemple, que vous ayez créé un lecteur multimédia avec ActionScript 2.0. Ce lecteur multimédia charge des contenus divers également créés en ActionScript 2.0. Il est impossible de créer un contenu en ActionScript 3.0 et de le charger dans le lecteur multimédia. Vous devez effectuer une migration du lecteur vers ActionScript 3.0.

  Néanmoins, si vous créez un lecteur multimédia en ActionScript 3.0, il peut effectuer de simples chargements de votre contenu en ActionScript 2.0.

Le tableau ci-après récapitule les limites des versions précédentes de Flash Player relatives au chargement de nouveaux contenus et à l'exécution de code, ainsi que les restrictions liées à la programmation croisée entre les fichiers SWF écrits en différentes versions d'ActionScript.

Fonctionnalité prise en charge	**Flash Player 7**	**Flash Player 8**	**Flash Player 9 et 10**
Peut charger les fichiers SWF pour	version 7 et antérieure	version 8 et antérieure	version 9 (ou 10) et antérieure
Machine virtuelle incluse	AVM1	AVM1	AVM1 et AVM2
Exécute les fichiers SWF écrits en ActionScript	1.0 et 2.0	1.0 et 2.0	1.0, 2.0 et 3.0

Dans le tableau ci-dessous, « Fonctionnalité prise en charge » fait référence au contenu lisible dans Flash Player 9 ou une version ultérieure. Le contenu lisible dans Flash Player version 8 ou antérieure peut être chargé, affiché, exécuté et programmé avec ActionScript 1.0 et 2.0 uniquement.

Fonctionnalité prise en charge	**Contenu créé en ActionScript 1.0 et 2.0**	**Contenu créé en ActionScript 3.0**
Peut charger du contenu et exécuter le code de contenus créé dans	ActionScript 1.0 et 2.0 uniquement	ActionScript 1.0 et 2.0 et ActionScript 3.0
Peut programmer le contenu créé dans	ActionScript 1.0 et 2.0 uniquement (ActionScript 3.0 via LocalConnection)	ActionScript 1.0 et 2.0 via LocalConnection ActionScript 3.0

Chapitre 3 : Prise en main d'ActionScript

Ce chapitre vise à vous expliquer les principes de base de la programmation avec ActionScript et à vous offrir les informations nécessaires à la compréhension des concepts et des exemples présentés dans le reste de ce manuel. Nous commencerons par évoquer les concepts de base de la programmation, tels qu'ils s'appliquent à ActionScript. Nous verrons aussi les principes fondamentaux qui régissent l'organisation et la construction d'une application ActionScript.

Concepts de programmation de base

ActionScript étant un langage de programmation, il vous sera plus facile de l'apprendre si vous maîtrisez déjà quelques concepts généraux de programmation.

Quel est le rôle d'un programme informatique ?

Pour commencer, il est intéressant d'avoir une idée conceptuelle de la nature et du rôle d'un programme informatique. Celui-ci présente deux aspects principaux :

- Il est constitué d'une série d'instructions ou d'étapes que l'ordinateur doit effectuer.
- Chaque étape implique à terme la manipulation d'informations ou de données.

En fait, un programme informatique n'est rien d'autre qu'une liste d'actions que vous demandez à l'ordinateur d'exécuter l'une après l'autre. Chacune de ces demandes d'exécution d'action s'appelle une *instruction*. Comme vous le verrez tout au long de ce manuel, dans ActionScript, chaque instruction se termine par un point-virgule.

Par nature, le seul rôle d'une instruction de programme consiste à manipuler quelques données stockées dans la mémoire de l'ordinateur. Voici un exemple simple : vous pouvez demander à l'ordinateur d'ajouter deux nombres et de stocker le résultat dans sa mémoire. Dans un cas de figure plus compliqué, imaginez un rectangle dessiné sur l'écran ; vous rédigez un programme pour le déplacer à un autre emplacement de l'écran. L'ordinateur garde une trace de certaines informations relatives au rectangle : les coordonnées x, y de sa position, sa largeur et sa longueur, sa couleur, etc. Chacune de ces informations est stockée dans la mémoire de l'ordinateur. Un programme qui déplacerait le rectangle vers un autre emplacement comprendrait des procédures du type « remplacer la coordonnée x par 200, remplacer la coordonnée y par 150 », c'est-à-dire qui spécifieraient de nouvelles valeurs pour les coordonnées x et y. Bien entendu, l'ordinateur agit sur ces données pour effectivement modifier l'image à l'écran en fonction de ces chiffres. Cependant, pour ce qui nous intéresse, il est suffisant de savoir que le déplacement d'un rectangle sur l'écran n'implique réellement que la modification de quelques bits de données dans la mémoire de l'ordinateur.

Variables et constantes

La programmation reposant sur la modification d'informations dans la mémoire de l'ordinateur, il est nécessaire d'établir une représentation de chaque information au sein du programme. Une *variable* est un nom qui représente une valeur dans la mémoire de l'ordinateur. Lorsque l'on rédige des instructions visant à manipuler des valeurs, on écrit le nom de la variable plutôt que la valeur. Chaque fois que l'ordinateur rencontre le nom de la variable dans le programme, il cherche dans sa mémoire la valeur à utiliser. Si vous disposez par exemple de deux variables appelées `value1` et `value2`, chacune contenant un nombre, vous pouvez additionner ces deux nombres à l'aide de l'instruction suivante :

```
value1 + value2
```

Lorsqu'il exécute véritablement la procédure, l'ordinateur recherche les valeurs correspondant à chaque variable et les ajoute.

Dans ActionScript 3.0, une variable se compose de trois éléments :

- Le nom de la variable
- Le type de données qui peut être stocké dans la variable
- La valeur réelle stockée dans la mémoire de l'ordinateur

Nous venons de voir comment l'ordinateur utilise le nom comme une balise d'emplacement destinée à la valeur. Le type de données a également une importance. Lorsque vous créez une variable dans ActionScript, vous spécifiez le type de données qu'elle contiendra. A partir de là, les instructions de votre programme peuvent uniquement stocker ce type de données dans la variable considérée et la valeur peut être manipulée selon les caractéristiques associées à ce type de données. Dans ActionScript, la création d'une variable (on parle également de *déclaration* de variable) s'effectue à l'aide de l'instruction `var`:

```
var value1:Number;
```

Dans ce cas, nous indiquons à l'ordinateur qu'il doit créer une variable `value1` qui contiendra uniquement des données Number, un type de données spécialement défini dans ActionScript. Il est possible de stocker immédiatement une valeur dans la variable :

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

Adobe Flash CS4 Professional propose une autre méthode de déclaration des variables. Lorsque vous placez un symbole de clip, un symbole de bouton ou un champ de texte sur la scène, vous pouvez lui attribuer un nom d'occurrence dans l'Inspecteur des Propriétés. En arrière-plan, Flash crée une variable du même nom que celui de l'occurrence, que vous pouvez utiliser dans votre code ActionScript pour faire référence à l'élément de la scène. Par exemple, si un symbole de clip se trouve sur la scène et que vous lui attribuez le nom d'occurrence `rocketShip`, chaque fois que vous utilisez la variable `rocketShip` dans votre code ActionScript, c'est en fait ce clip que vous manipulez.

Une constante s'apparente à une variable dans la mesure où elle correspond à un nom représentant une valeur dans la mémoire de l'ordinateur et est associée à un type de données spécifique. En revanche, vous ne pouvez affecter qu'une seule valeur à une constante dans une application ActionScript. Une fois affectée à une constante, une valeur reste la même dans l'ensemble de l'application. La syntaxe de déclaration d'une constante est identique à celle de la déclaration d'une variable, à la différence près que vous substituez le mot-clé `const` au mot-clé `var`.

```
const SALES_TAX_RATE:Number = 0.07;
```

Une constante permet de définir une valeur qui s'utilise à plusieurs emplacements dans un projet et reste identique dans des circonstances normales. L'utilisation d'une constante plutôt que d'une valeur littérale améliore la lisibilité de votre code. Il est plus facile, par exemple, de comprendre la finalité d'une ligne de code qui multiplie un prix par `SALES_TAX_RATE`, plutôt que par `0,07`. En outre, si vous représentez une constante par une valeur dans l'ensemble de votre projet et qu'il s'avère nécessaire de modifier cette valeur, il vous suffit d'intervenir à un seul emplacement (dans la déclaration de la constante), alors que si vous utilisez des valeurs littérales codées en dur, vous devez changer chaque occurrence.

Types de données

Dans ActionScript, de nombreux types de données sont à votre disposition pour la création de variables. Certains d'entre eux peuvent être considérés comme simples ou fondamentaux :

- String : une valeur textuelle, telle qu'un nom ou le texte d'un chapitre de livre
- Numeric : ActionScript 3.0 inclut trois types de données spécifiques aux valeurs numériques :
  - Number : toute valeur numérique, y compris les valeurs avec ou sans fraction

  - int : un nombre entier (sans fraction)
  - uint : un nombre entier « non signé », c'est-à-dire, un nombre entier qui ne peut pas être négatif
- Boolean : une valeur vrai/faux, qui indique par exemple si une instruction Switch est active ou si deux valeurs sont égales

Les types de données simples représentent une information unique : un nombre ou une séquence de texte, par exemple. Cependant, la majorité des types de données définis dans ActionScript peuvent être considérés comme complexes puisqu'ils représentent un ensemble de valeurs regroupées. Par exemple, une variable du type Date représente une valeur unique, un point temporel. Toutefois, la valeur date se compose en réalité de plusieurs valeurs (le jour, le mois, l'année, les heures, les minutes, les secondes, etc.), qui correspondent elles-mêmes à des nombres individuels. Ainsi, bien que nous percevions une date comme une valeur unique (et qu'il soit possible de la traiter comme telle en créant une variable Date), l'ordinateur, en interne, la considère comme un groupe de valeurs qui, ensemble, définissent une seule date.

La plupart des types de données intégrés, y compris ceux définis par les programmeurs, sont des types de données complexes. Voici quelques exemples de types de données complexes que vous pourriez rencontrer :

- MovieClip : un symbole de clip
- TextField : un champ de texte dynamique ou saisi
- SimpleButton : un symbole de bouton
- Date : une information relative à un point temporel (date et heure)

Deux termes sont souvent utilisés comme synonymes de type de données : classe et objet. Une *classe* est tout simplement la définition d'un type de données ; un modèle, en quelque sorte, s'appliquant à tous les objets du type de données et qui revient à dire « toutes les variables du type de données Exemple sont dotées des caractéristiques suivantes : A, B et C ». Un *objet*, quant à lui, est une occurrence réelle d'une classe. Une variable dont le type de données correspond à MovieClip, par exemple, peut être décrite comme un objet MovieClip. Les exemples ci-après décrivent la même chose :

- Le type de données de la variable `myVariable` est Number.
- La variable `myVariable` est une occurrence de Number.
- La variable `myVariable` est un objet Number.
- La variable `myVariable` est une occurrence de la classe Number.

Utilisation des objets

ActionScript constitue ce que l'on appelle un langage de programmation orienté objet. Ce terme désigne une simple approche de la programmation, rien d'autre qu'une manière d'organiser le code d'un programme à l'aide d'objets.

Nous avons défini plus haut un programme informatique comme une série de procédures ou d'instructions que l'ordinateur effectue. Nous pouvons alors considérer qu'un programme informatique n'est qu'une longue liste d'instructions. Dans le cas de la programmation orientée objets, les instructions du programme se divisent néanmoins en différents objets : le code étant rassemblé en groupes de fonctionnalités, un même conteneur réunit des types de fonctionnalités et des informations connexes.

En fait, si vous avez travaillé avec des symboles dans Flash, vous savez déjà manipuler les objets. Imaginons que vous ayez défini un symbole de clip (le dessin d'un rectangle, par exemple) et vous en ayez placé une copie sur la scène. Ce symbole de clip constitue aussi un objet (au sens littéral) dans ActionScript ; il s'agit d'une occurrence de la classe MovieClip.

Vous avez la possibilité de modifier plusieurs caractéristiques du clip. S'il est sélectionné, vous pouvez par exemple modifier certaines valeurs de l'Inspecteur des Propriétés, telles que la coordonnée x ou la largeur, ou encore différents réglages de couleur comme la transparence alpha ou l'application d'un filtre d'ombre portée. D'autres outils Flash vous permettent d'effectuer davantage de modifications, par exemple l'outil Transformation libre pour faire pivoter le rectangle. Toutes ces actions de transformation du symbole de clip disponibles dans l'environnement de programmation de Flash sont également disponibles dans ActionScript. Pour les utiliser, vous devez modifier les données réunies dans un ensemble appelé objet MovieClip.

Dans la programmation orientée objets que propose ActionScript, chaque classe comprend trois types de caractéristiques :

- Propriétés
- Méthodes
- Evénements

Pris ensemble, ces éléments servent à gérer les données que le programme utilise, et à déterminer les actions à exécuter ainsi que l'ordre d'exécution.

Propriétés

Une propriété représente l'une des données réunies dans un objet. Un objet song, par exemple, peut présenter des propriétés appelées `artist` et `title`. La classe MovieClip possède des propriétés telles que `rotation`, `x`, `width` et `alpha`. Les propriétés s'utilisent comme des variables individuelles; on pourrait même envisager les propriétés comme les variables enfant d'un objet.

Voici des exemples de code ActionScript utilisant des propriétés. Cette ligne de code déplace l'objet MovieClip nommé `square` vers la coordonnée x de 100 pixels :

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

Le code ci-après utilise la propriété rotation pour faire pivoter le MovieClip `square` de manière à lui donner la même orientation que le MovieClip `triangle` :

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

Ce code modifie l'échelle horizontale du MovieClip `square` pour qu'il soit une fois et demie plus large que précédemment :

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

Observez la structure commune : vous utilisez une variable (`square`, `triangle`) comme nom de l'objet, suivi d'un point() puis du nom de la propriété (`x`, `rotation`, `scaleX`). Le point, ou *opérateur point*, sert à indiquer que vous accédez à l'un des éléments enfant d'un objet. La structure dans son ensemble, « nom de variable-point-nom de propriété » est utilisée telle une variable, comme le nom d'une valeur unique dans la mémoire de l'ordinateur.

Méthodes

Une *méthode* est une action qui peut être effectuée par un objet. Par exemple, si vous avez élaboré dans Flash un symbole de clip dont le scénario contient plusieurs images clés et animations, ce clip peut être lu ou arrêté, ou recevoir l'instruction de placer la tête de lecture sur une image donnée.

Le code ci-dessous indique au MovieClip nommé `shortFilm` de commencer la lecture :

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

Cette ligne de code arrête la lecture du MovieClip `shortFilm` (la tête de lecture s'arrête à l'endroit où elle se trouve, comme lorsque vous mettez une vidéo en pause) :

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

Ce code-ci indique au MovieClip `shortFilm` de placer la tête de lecture sur Frame 1 et d'arrêter la lecture (comme si vous rembobiniez une vidéo) :

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

Comme vous pouvez le constater, l'accès aux méthodes, comme pour les propriétés, s'effectue en écrivant le nom de l'objet (une variable) suivi d'un point puis du nom de la méthode et de parenthèses. Les parenthèses servent à indiquer que vous *appelez* la méthode, c'est-à-dire que vous demandez à l'objet d'effectuer une action. Il arrive que des valeurs (ou variables) soient placées dans ces parenthèses de manière à transmettre des informations supplémentaires nécessaires à l'exécution de l'action. On appelle ces valeurs des *paramètres* de méthode. Par exemple, la méthode `gotoAndStop()` doit savoir quelle image atteindre ; un paramètre est donc requis dans les parenthèses. D'autres méthodes, telles `play()` et `stop()`, ont une signification univoque et n'ont donc besoin d'aucune information complémentaire. Elles sont néanmoins suivies de parenthèses.

Contrairement aux propriétés (et aux variables), les méthodes ne servent pas d'espaces réservés. Certaines méthodes peuvent cependant effectuer des calculs et renvoyer des résultats pouvant servir de variables. C'est le cas de la méthode `toString()` de la classe Number, qui convertit une valeur numérique en une représentation textuelle :

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

Par exemple, vous pouvez utiliser la méthode `toString()` si vous souhaitez afficher la valeur d'une variable Number dans un champ de texte à l'écran. La propriété `text` de la classe TextField (qui représente le contenu textuel réel affiché à l'écran) se définit comme une chaîne (String), de sorte qu'elle ne puisse contenir que des valeurs textuelles. Cette ligne de code convertit la valeur numérique de la variable `numericData` en texte, puis la fait apparaître à l'écran dans l'objet TextField nommé `calculatorDisplay` :

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

Evénements

Tel que nous l'avons décrit, un programme informatique est une série d'instructions que l'ordinateur exécute l'une après l'autre. Certains programmes très simples ne sont rien de plus : l'ordinateur exécute quelques procédures, puis le programme se termine. Toutefois, les programmes ActionScript sont conçus pour poursuivre leur exécution, dans l'attente d'une saisie utilisateur, par exemple. Les événements sont des mécanismes qui déterminent quelles instructions l'ordinateur doit exécuter et à quel moment.

Par essence, les *événements* sont des faits qui surviennent et auxquels ActionScript peut répondre parce qu'il en est conscient au moment où ils se produisent. De nombreux événements sont liés à l'interaction de l'utilisateur, par exemple un clic sur un bouton ou une pression sur une touche du clavier. Il existe néanmoins d'autres types d'événement. Par exemple, si vous utilisez ActionScript pour charger une image externe, un événement est capable de vous prévenir lorsque le chargement de l'image est terminé. Pendant l'exécution d'un programme ActionScript, Adobe Flash Player et Adobe AIR attendent que des événements se produisent et lorsque c'est le cas, ils exécutent le code ActionScript que vous avez spécifié en réponse.

Gestion des événements de base

La *gestion des événements* est la technique qui permet de spécifier les actions à exécuter en réponse à des événements particuliers. Lors de l'écriture de code ActionScript en vue de la gestion des événements, trois éléments importants sont à identifier :

- Source de l'événement : quel objet sera-t-il concerné par l'événement ? Par exemple, sur quel bouton aura lieu le clic ou quel est l'objet Loader qui charge l'image ? La source de l'événement est également appelée *cible de l'événement* car elle représente l'objet où l'événement est ciblé par Flash Player ou AIR (là où l'événement a lieu).

- Evénement : que doit-il se passer, à quel événement voulez-vous répondre? Ce point est très important, car de nombreux objets déclenchent plusieurs événements.
- Réponse : quelles actions doivent être exécutées lorsque l'événement se produit ?

Tout code ActionScript de gestion des événements doit contenir ces trois éléments et respecter la structure de base suivante (les éléments en gras sont des espaces réservés à remplir selon le cas envisagé) :

```
function eventResponse(eventObject:EventType):void
{
    // Actions performed in response to the event go here.
}

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

Ce code a un double rôle. Tout d'abord, il définit une fonction, qui est une manière de spécifier les actions à exécuter en réponse à l'événement. Ensuite, il appelle la méthode `addEventListener()` de l'objet source, « inscrivant » ainsi la fonction auprès de l'événement spécifié de manière que, dès que l'événement survient, les actions de la fonction aient lieu. Nous allons étudier chacun de ces rôles en détail.

Une *fonction* sert à regrouper des actions sous un nom unique, un raccourci qui vous permet de les exécuter. La fonction est identique à la méthode, à cette exception près qu'elle n'est pas nécessairement associée à une classe particulière (on pourrait d'ailleurs définir la méthode ainsi : une fonction associée à une classe donnée). Lorsque vous créez une fonction de gestion des événements, vous devez choisir le nom de la fonction (dans ce cas `eventResponse`) mais aussi spécifier un paramètre (`eventObject` dans cet exemple). La spécification d'un paramètre de fonction ressemble à la déclaration de variable ; vous devez dans ce cas aussi indiquer le type de données du paramètre (`EventType`, dans cet exemple).

Chaque type d'événement que vous souhaitez écouter est associé à une classe ActionScript. Le type de données que vous spécifiez pour le paramètre de fonction correspond systématiquement à la classe associée à l'événement auquel vous souhaitez réagir. Par exemple, un événement `click` (déclenché par un clic de souris sur un élément) est associé à la classe MouseEvent. Pour écrire une fonction d'écouteur pour un événement `click`, vous lui associez un paramètre de type MouseEvent. Enfin, entre les accolades d'ouverture et de fermeture (`{ ... }`), vous placez les instructions que l'ordinateur doit exécuter lorsque l'événement a lieu.

Une fois que vous avez écrit la fonction de gestion de l'événement, vous devez indiquer à l'objet source (celui qui provoque l'événement, par exemple le bouton) que cette fonction doit être appelée lorsque l'événement survient. Pour ce faire, appelez la méthode `addEventListener()` de cet objet (tous les objets liés à des événements ont une méthode `addEventListener()`). La méthode `addEventListener()` réclame deux paramètres :

- Tout d'abord, le nom de l'événement auquel vous voulez répondre. Comme nous l'avons déjà vu, chaque événement est affilié à une classe spécifique pour laquelle est définie une valeur propre à chacun d'eux (en quelque sorte le nom unique de l'événement). Cette valeur sert de premier paramètre.
- Vient ensuite le nom de votre fonction de réponse à l'événement. Sachez qu'un nom de fonction est écrit sans parenthèses lors du transfert en tant que paramètre.

Description du processus de gestion des événements

Vous trouverez ci-dessous une description détaillée du processus ayant lieu lorsque vous créez un écouteur d'événement. Dans ce cas, il s'agit d'un exemple illustrant la création d'une fonction d'écouteur appelée lorsque vous cliquez sur un objet `myButton`.

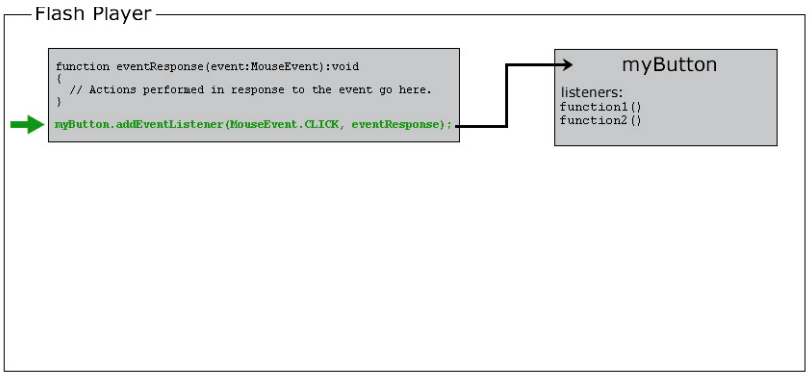
Le code écrit par le programmeur est le suivant :

```
function eventResponse(event:MouseEvent):void
{
    // Actions performed in response to the event go here.
}

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

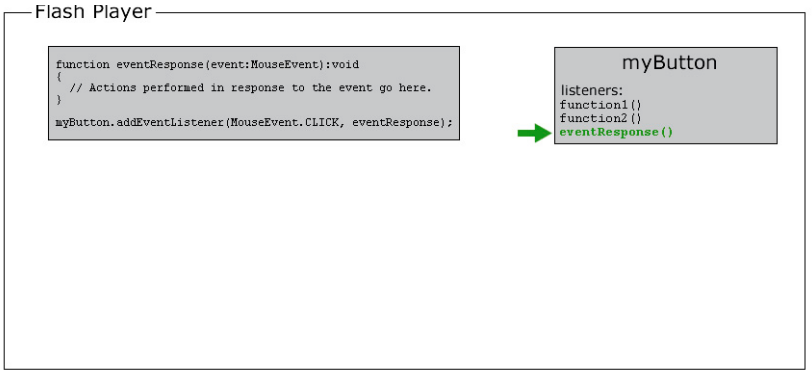
Voici comment ce code devrait fonctionner lorsqu'il est exécuté dans Flash Player. (Ce comportement s'applique également à Adobe AIR.) :

1 Lors du chargement du fichier SWF, Flash Player remarque qu'il existe une fonction `eventResponse()`.

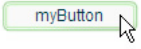
2 Flash Player exécute ensuite le code (notamment les lignes de code qui ne sont pas dans une fonction). Dans ce cas, il s'agit d'une seule ligne de code : l'appel de la méthode `addEventListener()` sur l'objet source de l'événement (`myButton`) et la transmission de la fonction `eventResponse` en tant que paramètre.



a En interne, `myButton` a une liste des fonctions qui écoutent chaque événement. Par conséquent, lorsque sa méthode `addEventListener()` est appelée, `myButton` stocke la fonction `eventResponse()` dans sa liste d'écouteurs d'événement.

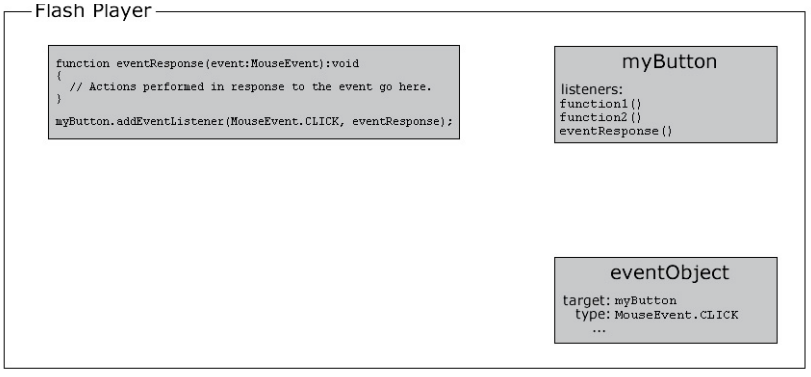


3 A un certain moment, l'utilisateur clique sur l'objet `myButton` et déclenche ainsi son événement `click` (identifié comme `MouseEvent.CLICK` dans le code).

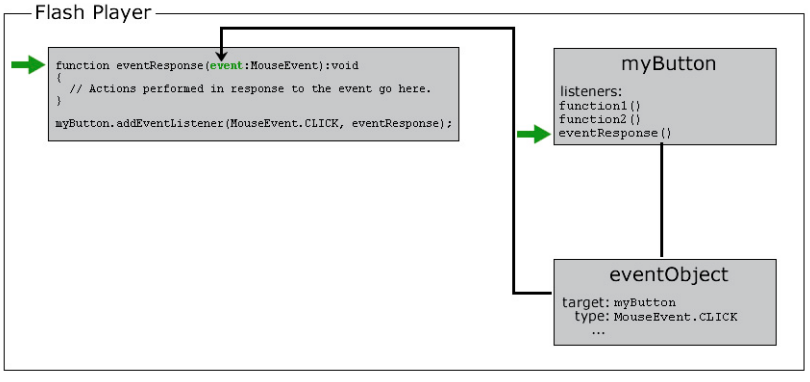


A ce stade, les opérations suivantes ont lieu :

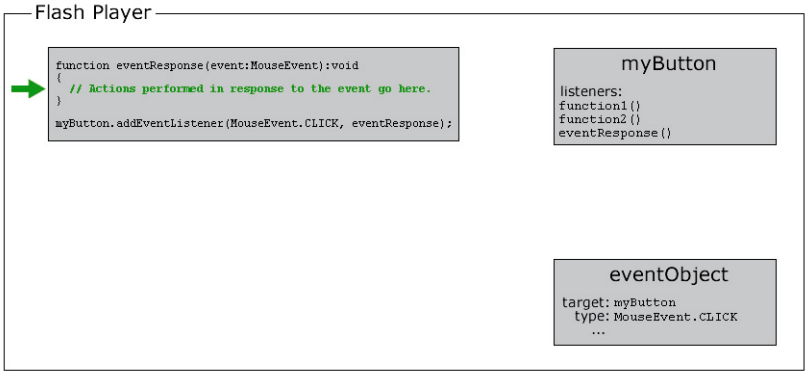
a Flash Player crée un objet, une occurrence de la classe associée à l'événement en question (MouseEvent dans cet exemple). Pour de nombreux événements, il s'agit d'une occurrence de la classe Event ; pour des événements de souris, une occurrence de MouseEvent et pour d'autres événements, une occurrence de la classe associée à cet événement. L'objet créé est appelé l'*objet événement*. Il contient des informations spécifiques sur l'événement qui s'est produit : son type, l'emplacement où il a eu lieu et toute autre information pertinente.



b Flash Player consulte ensuite la liste des écouteurs d'événement stockés par `myButton`. Il parcourt ces fonctions l'une après l'autre en les appelant et en transmettant l'objet événement à la fonction en tant que paramètre. Etant donné que la fonction `eventResponse()` est l'un des écouteurs de `myButton`, Flash Player appelle la fonction `eventResponse()` dans le cadre de ce processus.



c Lorsque la fonction `eventResponse()` est appelée, le code qu'elle contient est exécuté et vos actions spécifiées sont effectuées.



Exemples de gestion d'événements

Voici quelques exemples plus concrets d'événements qui vous donneront une idée des éléments les plus courants et des variations que vous pourrez utiliser lors de l'écriture de votre propre code de gestion des événements :

- Clic sur un bouton pour lancer la lecture du clip actif. Dans l'exemple suivant, `playButton` est le nom d'occurrence du bouton et `this` est un nom spécial qui signifie « l'objet actif» :

```
this.stop();

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

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

- Détection de la saisie dans un champ de texte. Dans cet exemple, `entryText` est un champ de saisie de texte et `outputText` est un champ de texte dynamique :

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

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

- Clic sur un bouton pour atteindre une URL. Dans ce cas, `linkButton` est le nom d'occurrence du bouton :

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

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

Création d'occurrences d'objets

Bien entendu, pour utiliser un objet dans ActionScript, cet objet doit exister. La création d'un objet repose en partie sur la déclaration d'une variable. Toutefois, celle-ci crée uniquement un emplacement vide dans la mémoire de l'ordinateur. Vous devez attribuer une valeur réelle à cette variable (c'est-à-dire créer un objet et le stocker dans la variable) avant de pouvoir l'utiliser ou la manipuler. Le processus de création d'un objet est appelé *instanciation* de l'objet, soit la création d'une occurrence d'une classe particulière.

La création d'une occurrence d'objet peut passer par une méthode simple qui n'implique pas ActionScript. Dans Flash, lorsque vous placez un symbole de clip, un symbole de bouton ou un champ de texte sur la scène, puis que vous lui attribuez un nom d'occurrence dans l'Inspecteur des Propriétés, l'application déclare une variable dotée de ce nom d'occurrence, crée une occurrence d'objet et stocke cet objet dans la variable. Il en va de même dans Adobe Flex Builder : si vous créez un composant MXML (soit par codage d'une balise MXML, soit en plaçant le composant dans l'éditeur en mode Création) et lui attribuez un identifiant (dans la balise MXML ou dans la vue des propriétés Flex), cet identifiant devient le nom d'une variable ActionScript. A ce moment-là, une occurrence du composant est créée et stockée dans la variable.

Toutefois, vous ne pourrez pas toujours créer un objet visuellement. Plusieurs méthodes permettent aussi de créer des occurrences d'objet à l'aide d'ActionScript uniquement. A partir de plusieurs types de données ActionScript, vous pouvez tout d'abord créer une occurrence en utilisant une *expression littérale*, une valeur écrite directement dans le code ActionScript. Voici quelques exemples :

- Valeur numérique littérale (entrez le nombre directement) :

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

- Valeur de chaîne littérale (entourez le texte de doubles guillemets) :

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

- Valeur booléenne littérale (utilisez la valeur littérale `true` ou `false`) :

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

- Valeur de tableau littérale (placez entre crochets une liste de valeurs séparées par des virgules) :

```
var seasons:Array = ["spring", "summer", "autumn", "winter"];
```

- Valeur XML littérale (entrez le XML directement) :

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

ActionScript définit également des expressions littérales pour les types de données Array, RegExp, Object et Function. Pour plus de détails sur ces classes, consultez les chapitres « Utilisation de tableaux » à la page 158 et « Utilisation d'expressions régulières » à la page 211, ainsi que la section « Type de données Object » à la page 60.

Pour tout autre type de données, créez une occurrence d'objet à l'aide de l'opérateur `new` et du nom de classe, comme suit :

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

Pour cette méthode de création d'un objet à l'aide de l'opérateur `new`, on parle souvent « d'appeler le constructeur de la classe ». Un *constructeur* est une méthode spéciale qui est appelée dans le cadre de la création d'une occurrence de classe. Notez que lorsque vous créez une occurrence ainsi, vous placez des parenthèses après le nom de classe et spécifiez parfois des valeurs de paramètres, comme vous le faites lorsque vous appelez une méthode.

Remarquez que vous pouvez utiliser l'opérateur `new` pour créer une occurrence d'objet, même pour les types de données qui permettent de créer des occurrences avec une expression littérale. Par exemple, ces deux lignes de code produisent exactement le même résultat :

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

Il est important de se familiariser avec la méthode `new` *ClassName*`()` de création d'objets. Dans le cas d'un type de données ActionScript qui ne dispose pas de représentation visuelle (et ne peut donc pas être créé en plaçant un élément sur la scène de Flash ou dans l'éditeur en mode Création de Flex Builder), vous pouvez uniquement créer un objet directement dans ActionScript à l'aide de l'opérateur `new`.

Dans Flash, l'opérateur `new` peut en outre servir à créer une occurrence d'un symbole de clip défini dans la bibliothèque sans être placé sur la scène. Pour plus d'informations à ce sujet, consultez la section « Création d'objets MovieClip à l'aide d'ActionScript » à la page 421.

Eléments de programme courants

Outre la déclaration des variables, la création d'occurrences d'objets et la manipulation d'objets à l'aide des propriétés et des méthodes, d'autres éléments de construction peuvent servir à la création d'un programme ActionScript.

Opérateurs

Les *opérateurs* sont des symboles spéciaux (et parfois des mots) qui permettent d'effectuer des calculs. Ils sont surtout utilisés dans les opérations mathématiques et la comparaison de valeurs. En règle générale, un opérateur utilise une ou plusieurs valeurs et « établit » un résultat unique. Par exemple :

- L'opérateur de somme (`+`) ajoute deux valeurs pour obtenir un nombre unique :

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

- L'opérateur de multiplication (`*`) multiplie une valeur par une autre pour obtenir un nombre unique :

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

- L'opérateur d'égalité (`==`) compare deux valeurs pour vérifier si elles sont égales, afin d'obtenir une valeur vrai/faux (booléenne) unique :

  ```
  if (dayOfWeek == "Wednesday")
  {
      takeOutTrash();
  }
  ```

  Comme l'indique la figure ci-dessus, l'opérateur d'égalité et les autres opérateurs de comparaison sont le plus souvent utilisés avec l'instruction `if` afin de déterminer si certaines actions sont à effectuer ou non.

Pour plus d'informations et d'exemples sur l'utilisation des opérateurs, consultez la section « Opérateurs » à la page 70.

Commentaires

Lors de la rédaction du code ActionScript, il peut s'avérer utile de conserver des notes personnelles, expliquant par exemple le fonctionnement de certaines lignes de code ou la raison pour laquelle vous avez fait tel ou tel choix. Les *commentaires de code* vous permettent d'insérer du texte dans votre code, que l'ordinateur devra ignorer. ActionScript comprend deux types de commentaires :

- Commentaire sur une ligne : un commentaire sur une ligne est signalé par l'insertion de deux barres obliques en un emplacement quelconque d'une ligne. Tout ce qui apparaît après les barres obliques et jusqu'à la fin de la ligne est ignoré par l'ordinateur :

```
// This is a comment; it's ignored by the computer.
var age:Number = 10; // Set the age to 10 by default.
```

- Commentaire sur plusieurs lignes : un commentaire sur plusieurs lignes comprend un marqueur de début (`/*`), le commentaire lui-même, puis un marqueur de fin de commentaire (`*/`). L'ordinateur ignore tout ce qui apparaît entre les marqueurs de début et de fin, quel que soit le nombre de lignes utilisé par le commentaire :

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

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

Une autre utilité des commentaires est de « désactiver » temporairement une ou plusieurs lignes de code ; c'est le cas, par exemple, si vous testez différentes façons d'aboutir à un résultat ou essayez d'identifier pourquoi le code ActionScript ne fonctionne pas comme vous le pensiez.

Contrôle du flux

Dans bien des cas, il vous sera nécessaire de répéter des actions de votre code, d'en effectuer certaines et pas d'autres, de réaliser des actions de remplacement selon les conditions rencontrées, etc. Le *contrôle de flux* permet de maîtriser les actions exécutées. ActionScript propose plusieurs types d'éléments de contrôle de flux.

- Fonctions : les fonctions sont comme des raccourcis, elles permettent de regrouper sous un même nom une série d'actions qui serviront à des calculs. Essentielles à la gestion des événements, elles constituent en outre un outil générique de regroupement des instructions. Pour plus d'informations sur les fonctions, consultez la section « Fonctions » à la page 81.
- Boucles : les structures en boucle vous permettent de désigner un jeu d'instructions que l'ordinateur exécutera un nombre défini de fois ou jusqu'à ce qu'une condition change. Souvent, les boucles sont utilisées pour manipuler plusieurs éléments connexes à l'aide d'une variable dont la valeur change à chaque fois que l'ordinateur achève une boucle. Pour plus d'informations sur les boucles, consultez la section « Boucle » à la page 78.
- Instructions conditionnelles : les instructions conditionnelles permettent de désigner certaines actions à effectuer uniquement dans certaines circonstances ou de définir des ensembles d'actions destinés à différentes conditions. L'instruction conditionnelle la plus courante est `if`. Celle-ci vérifie la valeur ou l'expression placée dans ses parenthèses. Si le résultat est `true`, les lignes de code entre accolades sont exécutées ; dans le cas contraire, elles sont ignorées. Par exemple :

```
if (age < 20)
{
    // show special teenager-targeted content
}
```

L'instruction `if`, associée à l'instruction `else`, vous permet de désigner les actions à effectuer si la condition n'est pas `true` :

```
if (username == "admin")
{
    // do some administrator-only things, like showing extra options
}
else
{
    // do some non-administrator things
}
```

Pour plus d'informations sur les instructions conditionnelles, consultez la section « Instructions conditionnelles » à la page 76.

Exemple : élément de portfolio d'animation

Cet exemple indique comment vous pouvez assembler des éléments d'ActionScript dans une application complète ActionScript. L'élément de portfolio d'animation est un exemple de la façon dont vous pourriez ajouter à une animation linéaire existante (par exemple, un élément créé pour un client) des éléments interactifs mineurs pour l'incorporer dans un portfolio en ligne. Les éléments interactifs à ajouter sont deux boutons sur lesquels l'utilisateur peut cliquer : un pour lancer l'animation et un pour accéder à une URL distincte (telle que le menu du portfolio ou la page d'accueil de l'auteur).

Le processus de création de cet élément peut être divisé en quatre sections principales :

1. Préparer le fichier FLA pour ajouter des éléments ActionScript interactifs
2. Créer et ajouter les boutons
3. Ecrire le code ActionScript
4. Tester l'application

Préparation à l'ajout d'interactivité

Avant d'ajouter des éléments interactifs à notre animation, nous devons configurer le fichier FLA en créant des emplacements pour ajouter notre nouveau contenu. Ceci comprend la création d'un espace sur la scène où les boutons sont placés, et la création d'un espace dans le fichier FLA pour garder différents éléments séparés.

Pour configurer votre FLA et ajouter des éléments interactifs :

1. Si vous n'avez pas encore d'animation linéaire à laquelle vous ajouterez de l'interactivité, créez un fichier FLA avec une animation simple (une interpolation de mouvement simple ou une interpolation de forme, par exemple). Autrement, ouvrez le fichier FLA contenant l'animation que vous présentez dans le projet et enregistrez-le sous un nouveau nom pour créer un autre fichier de travail.
2. Choisissez l'endroit où vous souhaitez que les deux boutons apparaissent à l'écran (un pour lancer l'animation et l'autre pour effectuer un lien vers le portfolio de l'auteur ou la page d'accueil). Si nécessaire, libérez ou ajoutez de l'espace sur la scène pour ce nouveau contenu. Si l'animation n'en possède pas, vous pouvez créer une page de garde sur la première image (vous pouvez décaler l'animation afin qu'elle démarre sur l'image 2 ou ultérieurement).
3. Ajoutez un nouveau calque, au-dessus des autres dans le scénario, et renommez-le **buttons**. Il s'agit du calque auquel vous ajouterez les boutons.
4. Ajoutez un nouveau calque, au-dessus du calque buttons, et nommez-le **actions**. Il s'agit du calque auquel vous ajouterez le code ActionScript à votre application.

Création et ajout de boutons

Nous allons ensuite créer et positionner les boutons qui constitueront le centre de notre application interactive.

Pour créer et ajouter des boutons au fichier FLA :

1. A l'aide des outils de dessin, créez l'aspect visuel de votre premier bouton (celui de lecture) sur le calque buttons. Par exemple, vous pouvez dessiner un ovale horizontal avec du texte par-dessus.
2. A l'aide de l'outil de sélection, sélectionnez toutes les parties graphiques du bouton.
3. Dans le menu principal, choisissez Modifier > Convertir en symbole.
4. Dans la boîte de dialogue, choisissez le type de symbole de bouton, donnez-lui un nom et cliquez sur OK.
5. Le bouton étant sélectionné, dans l'Inspecteur des Propriétés, affectez-lui le nom d'occurrence **playButton**.
6. Répétez les étapes 1 à 5 afin de créer le bouton qui permettra à l'utilisateur d'accéder à la page d'accueil de l'auteur. Nommez ce bouton **homeButton**.

Ecriture du code

Le code ActionScript pour cette application peut être divisé en trois ensembles de fonctionnalités, même s'ils vont tous être entrés au même endroit. Le code doit effectuer les trois opérations suivantes :

- Arrêter la tête de lecture dès le chargement du fichier SWF (lorsque la tête de lecture atteint l'image 1).
- Ecouter un événement pour lancer la lecture du fichier SWF lorsque l'utilisateur clique sur le bouton de lecture.
- Ecouter un événement pour que le navigateur accède à l'URL appropriée lorsque l'utilisateur clique sur le bouton de la page d'accueil de l'auteur.

Pour créer un code qui arrête la tête de lecture lorsqu'elle atteint l'image 1 :

1. Sélectionnez l'image-clé sur l'image 1 du calque actions.
2. Pour ouvrir le panneau Actions, sélectionnez Fenêtre > Actions dans le menu principal.
3. Dans le panneau Script, entrez le code suivant :

```
stop();
```

Pour écrire un code qui lance l'animation lorsque l'utilisateur clique sur le bouton de lecture :

1. A la fin du code entré aux étapes précédentes, ajoutez deux lignes vides.
2. Entrez le code suivant en bas du script :

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

   Ce code définit une fonction appelée `startMovie()`. Lorsque la fonction `startMovie()` est appelée, elle lance la lecture du scénario principal.

3. Sur la ligne qui suit le code ajouté à l'étape précédente, entrez cette ligne de code :

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

   Cette ligne de code enregistre la fonction `startMovie()` comme écouteur pour l'événement `click` de `playButton`. Ainsi, chaque fois que l'utilisateur clique sur le bouton `playButton`, la fonction `startMovie()` est appelée.

Pour rédiger un code qui permet au navigateur d'accéder à une URL lorsque l'utilisateur clique sur le bouton de la page d'accueil :

1 A la fin du code entré aux étapes précédentes, ajoutez deux lignes vides.

2 Entrez ce code au bas du script :

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

Ce code définit une fonction `gotoAuthorPage()`. Cette fonction crée d'abord une occurrence de URLRequest représentant l'URL http://example.com/, puis transmet cette URL à la fonction `navigateToURL()` afin que le navigateur de l'utilisateur l'ouvre.

3 Sur la ligne qui suit le code ajouté à l'étape précédente, entrez cette ligne de code :

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

Cette ligne de code enregistre la fonction `gotoAuthorPage()` comme écouteur pour l'événement `click` de `homeButton`. Ainsi, chaque fois que l'utilisateur clique sur le bouton `homeButton`, la fonction `gotoAuthorPage()` est appelée.

Test de l'application

A ce stade, l'application devrait fonctionner complètement. Testons-la pour nous assurer que c'est le cas.

Pour tester l'application :

1 Dans le menu principal, sélectionnez Contrôle > Tester l'animation. Flash crée le fichier SWF et l'ouvre dans une fenêtre Flash Player.

2 Testez les deux boutons pour vérifier qu'ils fonctionnent.

3 Si ce n'est pas le cas, vérifiez les points suivants :

- Les deux boutons ont-ils des noms d'occurrence différents ?
- Les appels à la méthode `addEventListener()` utilisent-ils les mêmes noms que les noms d'occurrence des boutons ?
- Les noms d'événement corrects sont-ils utilisés dans les appels à la méthode `addEventListener()` ?
- Le paramètre correct est-il spécifié pour chacune des fonctions ? (Elles doivent toutes les deux avoir un seul paramètre avec le type de données MouseEvent.)

Tous ces points et la plupart des autres erreurs possibles devraient entraîner l'apparition d'un message lorsque vous choisissez la commande Tester l'animation ou lorsque vous cliquez sur le bouton. Recherchez les erreurs de compilation dans le panneau prévu à cet effet (celles qui ont lieu lorsque vous choisissez d'abord Tester l'animation), et recherchez les erreurs d'exécution dans le panneau Sortie (erreurs qui ont lieu pendant la lecture du fichier SWF - lorsque vous cliquez sur un bouton, par exemple).

Création d'applications avec ActionScript

La création d'une application avec ActionScript nécessite d'autres connaissances que la syntaxe et les noms de classes à utiliser. Bien que le contenu de ce manuel soit essentiellement axé sur ces deux sujets (la syntaxe et l'utilisation des classes ActionScript), d'autres informations pourront vous être utiles : notamment quels sont les programmes qui permettent d'écrire du code ActionScript ? comment ce code s'organise-t-il et s'intègre-t-il dans une application et quelles étapes faut-il respecter dans le développement d'une application ActionScript ?

Options d'organisation du code

Le code ActionScript 3.0 peut servir à générer de nombreuses applications, qu'il s'agisse d'une simple animation graphique ou d'un système complexe de traitement des transactions client/serveur. Selon le type d'application envisagé, vous choisirez l'une ou plusieurs des méthodes suivantes pour intégrer ActionScript dans votre projet.

Stockage du code dans les images d'un scénario Flash

Dans l'environnement de programmation Flash, vous pouvez ajouter du code ActionScript à toute image placée dans un scénario. Ce code sera exécuté pendant la lecture du clip, au moment où la tête de lecture atteindra l'image.

L'insertion de code ActionScript dans des images est une manière simple d'ajouter des comportements à des applications créées dans l'outil de programmation Flash. Vous pouvez placer du code dans n'importe quelle image du scénario principal ou de celui d'un symbole de clip. Cette souplesse a néanmoins un coût. Lorsque vous créez des applications assez volumineuses, vous risquez de ne plus savoir quelles images contiennent quels scripts. A terme, cela peut compliquer la maintenance de l'application.

Pour simplifier l'organisation de leur code ActionScript, les développeurs placent ce code uniquement dans la première image du scénario ou sur un calque spécifique du document Flash. Il est ainsi plus facile de localiser et de maintenir le code dans les fichiers FLA Flash. Toutefois, la réutilisation du même code dans un autre projet Flash oblige à copier et coller le code dans le nouveau fichier.

Si vous voulez continuer à pouvoir utiliser votre code ActionScript dans de futurs projets Flash, vous préférerez sans doute stocker ce code dans des fichiers ActionScript externes (des fichiers texte dotés de l'extension .as).

Stockage du code dans des fichiers ActionScript

Si votre projet implique une quantité importante de code ActionScript, la meilleure solution consiste à stocker le code dans des fichiers source ActionScript (des fichiers texte dotés de l'extension .as). Un fichier ActionScript peut suivre deux structures, selon l'utilisation que vous prévoyez d'en faire dans votre application.

- Code ActionScript non structuré : les lignes de code ActionScript, y compris les instructions et les définitions de fonction, sont écrites comme si elles étaient saisies directement dans un script de scénario, un fichier MXML, etc.

  Rédigé de cette façon, le code est accessible par le biais de l'instruction ActionScript `include` ou de la balise `<mx:Script>` dans Adobe Flex MXML. L'instruction ActionScript `include` provoque l'insertion du contenu d'un fichier ActionScript externe à un endroit particulier d'un script et sur une étendue donnée, comme s'il avait été saisi directement. Dans le langage Flex MXML, la balise `<mx:Script>` vous permet de spécifier l'attribut source qui identifie le fichier ActionScript externe à charger à cet endroit de l'application. Par exemple, la balise suivante charge un fichier ActionScript externe appelé Box.as :

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

- Définition d'une classe ActionScript : la définition d'une classe ActionScript ainsi que ses définitions de méthode et de propriété.

Lorsque vous définissez une classe, vous pouvez accéder au code ActionScript correspondant en créant une occurrence de cette classe et en utilisant ses propriétés, méthodes et événements, comme vous le feriez avec toute classe ActionScript intégrée. Cela implique deux opérations :

- Utilisez l'instruction `import` pour spécifier le nom complet de la classe, de manière à ce que le compilateur ActionScript sache où la trouver. Par exemple, pour utiliser la classe MovieClip dans ActionScript, vous devez commencer par importer cette classe à l'aide de son nom complet, en incluant le package et la classe.

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

  Une autre solution consiste à importer le package contenant la classe MovieClip, ce qui revient à écrire des instructions `import` pour chaque classe du package :

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

  Cette obligation d'importer les classes auxquelles vous faites référence dans votre code ne s'applique pas aux classes de niveau supérieur, qui ne sont pas définies dans le package.

  Remarque : *dans Flash, pour les scripts joints à des images du scénario, les classes intégrées (dans les packages flash.*) sont automatiquement importées. Toutefois, lorsque vous écrivez vos propres classes, que vous utilisez des composants de création Flash (les packages fl.*) ou que vous travaillez dans Flex, vous devez importer explicitement toute classe afin d'écrire du code qui crée les occurrences correspondantes.*

- Rédigez le code qui fait spécifiquement référence au nom de classe (normalement par la déclaration d'une variable dont le type de données est cette classe et par la création d'une occurrence de la classe à stocker dans la variable). Lorsque vous faites référence à un autre nom de classe dans le code ActionScript, vous indiquez au compilateur de charger la définition de cette classe. Par exemple, si l'on considère une classe externe appelée Box, l'instruction suivante provoque la création d'une nouvelle occurrence de la classe Box :

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

  Lorsque le compilateur rencontre pour la première fois la référence de la classe Box, il effectue une recherche dans le code source chargé afin de localiser la définition de la classe Box.

Choix de l'outil approprié

Selon les contraintes qu'impose votre projet et les ressources dont vous disposez, vous pouvez avoir le choix entre plusieurs outils (à utiliser individuellement ou en combinaison) pour l'écriture et la modification de votre code ActionScript.

Outil de programmation Flash

Outre ses capacités de création graphique et d'animation, Adobe Flash CS4 Professional comprend des outils qui permettent de manipuler le code ActionScript, qu'il soit joint à des éléments d'un fichier FLA ou regroupé dans des fichiers ActionScript externes. L'outil de programmation Flash s'avère idéal pour les projets impliquant des animations ou vidéos conséquentes, ou lorsque vous désirez créer la plupart des actifs graphiques vous-même, et tout particulièrement lorsque l'interaction utilisateur ou la fonctionnalité assurée par ActionScript est minimale. Cet outil peut également vous paraître adapté au développement de votre projet ActionScript si vous préférez créer les actifs visuels et écrire le code dans une seule et même application. Enfin, cet outil Flash peut vous convenir si vous souhaitez utiliser des composants d'interface préintégrés et que la réduction de la taille du fichier SWF et la facilité d'enveloppement visuel sont des aspects essentiels de votre projet.

Adobe Flash CS4 Professional inclut deux outils permettant l'écriture de code ActionScript :

- Panneau Actions : disponible lorsque vous manipulez un fichier FLA, ce panneau vous permet d'écrire du code ActionScript associé aux images d'un scénario.

- Fenêtre de script : la fenêtre de script est un éditeur de texte dédié permettant de travailler sur des fichiers de code ActionScript (.as).

Flex Builder

Adobe Flex Builder est le principal outil de création de projets avec la structure Flex. Au-delà de ses outils de présentation visuelle et d'édition MXML, Flex Builder comprend un éditeur ActionScript complet, qui permet de créer des projets Flex ou ActionScript. Les applications Flex présentent de nombreux avantages, notamment un large éventail de commandes d'interface préintégrées et de commandes de disposition dynamique souples, ainsi que des mécanismes intégrés permettant de manipuler des sources de données externes et de lier des données externes aux éléments d'interface utilisateur. Toutefois, ces fonctions nécessitant davantage de code, les applications Flex se caractérisent par une taille de fichier SWF supérieure et leur enveloppe ne peut être remaniée aussi facilement que leurs homologues Flash.

Utilisez Flex Builder si vous voulez créer avec Flex, dans un seul et même outil, des applications de données sur Internet riches en fonctions, tout en modifiant du code ActionScript et MXLM et en disposant les éléments de manière visuelle.

Editeur ActionScript tiers

Les fichiers ActionScript (.as) étant stockés comme de simples fichiers texte, tout programme susceptible de modifier des fichiers texte brut peut servir à écrire des fichiers ActionScript. Outre les produits ActionScript d'Adobe, plusieurs programmes tiers d'édition de texte ont été créés avec des fonctions propres à ActionScript. Vous pouvez écrire un fichier MXML ou des classes ActionScript à l'aide de tout éditeur de texte. A partir de ces fichiers, vous pouvez ensuite créer une application SWF (une application Flex ou ActionScript seul) à l'aide du kit SDK Flex, qui comprend les classes d'application Flex ainsi que le compilateur Flex. Pour de nombreux développeurs, une autre solution consiste à écrire les classes ActionScript dans un éditeur ActionScript tiers, en combinaison avec l'outil de programmation Flash pour la création du contenu graphique.

Vous pouvez choisir un éditeur ActionScript tiers dans les cas suivants :

- Vous préférez écrire le code ActionScript dans un programme distinct, tout en concevant les éléments visuels dans Flash.
- Vous utilisez une application de programmation non ActionScript (par exemple pour la création de pages HTML ou l'élaboration d'application dans un autre langage de programmation) et vous souhaitez l'utiliser pour le code ActionScript également.
- Vous voulez créer des projets ActionScript seul ou Flex à l'aide du kit SDK Flex sans avoir à acquérir Flash ou Flex Builder.

Les principaux éditeurs de code prenant en charge ActionScript sont les suivants :

- Adobe Dreamweaver® CS4
- ASDT
- FDT
- FlashDevelop
- PrimalScript
- SE|PY

Processus de développement ActionScript

Quelle que soit la taille de votre projet ActionScript, l'utilisation d'un processus de conception et de développement vous aidera à travailler plus efficacement. Les étapes ci-après forment le processus de développement de base pour la conception d'une application avec ActionScript 3.0 :

1 Concevez votre application.

   Avant de commencer à construire votre application, vous devez la décrire d'une manière ou d'une autre.

2 Composez votre code ActionScript 3.0.

   Vous pouvez créer du code ActionScript dans Flash, Flex Builder, Dreamweaver ou un éditeur de texte.

3 Créez un fichier application Flash ou Flex pour exécuter votre code.

   Dans l'outil de programmation Flash, cela implique la création d'un nouveau fichier FLA, la définition des paramètres de publication, l'ajout de composants d'interface à l'application et le référencement du code ActionScript. Dans l'environnement de développement Flex, la création d'un nouveau fichier d'application nécessite la définition de l'application et l'ajout de composants d'interface à l'aide de MXML, puis le référencement du code ActionScript.

4 Publiez et testez votre application ActionScript.

   Cela implique que vous exécutiez votre application au sein de l'environnement de programmation Flash ou de développement Flex, et que vous vérifiiez qu'elle effectue toutes les opérations voulues.

Il n'est pas indispensable de suivre ces étapes dans cet ordre ou d'achever l'une d'elles avant de passer à la suivante. Par exemple, vous pouvez concevoir un écran de votre application (étape 1), puis créer les graphiques, boutons, etc. (étape 3), avant d'écrire le code ActionScript (étape 2) et de le tester (étape 4). Vous pouvez tout aussi bien concevoir une partie de l'écran, puis ajouter un bouton ou un élément d'interface à la fois, écrire le code ActionScript correspondant et le tester dès qu'il est prêt. Bien qu'il soit judicieux de garder à l'esprit ces quatre stades du processus de développement, il est en pratique plus efficace d'aller et venir entre ces étapes en fonction des besoins.

Création de vos propres classes

Le processus de création des classes destinées à vos projets peut paraître rébarbatif. Cependant, la partie la plus difficile de la création d'une classe est sa conception, c'est-à-dire l'identification des méthodes, des propriétés et des événements qu'elle comprend.

Stratégies de conception d'une classe

La conception orientée objet est un sujet complexe ; des carrières entières ont été consacrées à l'étude académique et à la pratique professionnelle de cette discipline. Voici tout de même quelques suggestions d'approches qui vous aideront à lancer votre projet.

1 Réfléchissez au rôle que les occurrences de la classe auront à jouer dans l'application. En règle générale, les objets servent l'un des objectifs suivants :

   - Objet de valeur : ces objets constituent avant tout des conteneurs de données, c'est-à-dire qu'ils possèdent plusieurs propriétés et peu de méthodes (parfois aucune). Il s'agit en général d'une représentation dans le code d'éléments clairement définis, tels qu'une classe Song (représentant une seule chanson) ou une classe Playlist (représentant un groupe conceptuel de chansons) dans une application musicale.

- Objet d'affichage : ce type correspond à des objets qui s'affichent réellement à l'écran, par exemple des éléments d'interface, tels que des listes déroulantes ou des libellés d'état, des éléments graphiques tels que des créatures dans un jeu vidéo, etc.
- Structure d'application : ces objets jouent un large éventail de rôles dans la logique ou le traitement effectué par les applications. Il s'agit par exemple d'un objet réalisant des calculs dans une simulation de biologie, un objet chargé de synchroniser les valeurs entre une commande physique et le volume de sortie d'une application musicale, un objet que gèrent les règles d'un jeu vidéo ou un objet qui charge une image enregistrée dans une application de dessin.

2 Décidez de la fonctionnalité requise pour la classe. Les différentes fonctionnalités constituent souvent les méthodes de la classe.

3 Si la classe est destinée à servir d'objet de valeur, choisissez les données que les occurrences pourront inclure. Ces éléments peuvent facilement devenir des propriétés.

4 Puisque vous concevez la classe spécialement pour votre projet, le plus important est que vous établissiez la fonctionnalité nécessaire à votre application. Pour vous aider, vous pouvez répondre à ces questions :

- Quel type d'informations l'application stockera, surveillera et manipulera-t-elle ? Vous pourrez alors identifier les objets de valeurs et les propriétés qui vous serviront.
- Quels jeux d'actions devront être exécutés, par exemple lors du premier chargement de l'application, lorsque l'utilisateur clique sur un bouton donné, lorsque la lecture de la séquence s'arrête, etc. ? Ces éléments constituent souvent des méthodes (ou des propriétés, si les actions consistent uniquement à modifier des valeurs isolées).
- Pour chaque action considérée, quelles informations seront nécessaires à la classe pour son exécution ? Ces éléments deviennent les paramètres de la méthode.
- Pendant le fonctionnement de l'application, quelles modifications, qui devront être communiquées à d'autres parties de l'application, surviendront dans la classe ? Ces éléments forment en général des événements.

5 Si un objet existant est semblable à l'objet dont vous avez besoin mais qu'il lui manque certaines fonctionnalités que vous souhaitez ajouter, envisagez de créer une sous-classe (une classe qui repose sur la fonctionnalité d'une classe existante plutôt que de définir l'ensemble de sa fonctionnalité propre). Par exemple, si vous voulez créer une classe correspondant à un objet affiché à l'écran, vous pouvez créer votre classe en vous appuyant sur le comportement de l'un des objets d'affichage existants (par exemple, Sprite ou MovieClip). Dans ce cas, MovieClip (ou Sprite) constituerait la *classe de base*, que votre classe viendrait étendre. Pour plus d'informations sur la création d'une sous-classe, consultez la section « Héritage » à la page 111.

Ecriture du code d'une classe

Une fois que vous avez conçu votre classe ou au moins identifié les informations qu'elle manipulera et les actions qu'elle devra effectuer, l'écriture et la syntaxe à utiliser sont relativement simples.

Voici la procédure minimale de création d'une classe ActionScript :

1 Ouvrez un nouveau document texte dans un programme ActionScript tel que Flex Builder ou Flash, dans un outil de programmation générique tel que Dreamweaver ou dans toute application vous permettant de manipuler des documents en texte brut.

2 Saisissez une instruction `class` afin de définir le nom de la classe. Pour ce faire, entrez les mots `public class`, puis le nom de la classe suivi d'une paire d'accolades qui entoureront le contenu de la classe (les définitions de méthode et de propriété). Par exemple :

```
public class MyClass
{
}
```

Le mot `public` indique que la classe est accessible par tout autre code. Pour d'autres possibilités, consultez la section « Attributs d'espace de noms pour le contrôle d'accès » à la page 97.

3 Entrez une instruction `package`, qui indique le nom du package dans lequel votre classe apparaîtra. La syntaxe est le mot `package` suivi du nom complet du package, puis d'une paire d'accolades (qui entoureront l'élément structurel `class`). Par exemple, nous pouvons modifier le code précédent comme suit :

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

4 Définissez chaque propriété de la classe à l'aide de l'instruction `var` ajoutée dans le corps de la classe. Utilisez la même syntaxe que pour la déclaration de variable (en y ajoutant le qualificatif `public`). Par exemple, les lignes suivantes ajoutées entre les accolades de la définition de classe permettent de créer des propriétés appelées `textVariable`, `numericVariable` et `dateVariable` :

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

5 Définissez chaque méthode de la classe à l'aide de la syntaxe utilisée pour définir une fonction. Par exemple :

- Pour créer la méthode `myMethod()`, saisissez :

```
public function myMethod(param1:String, param2:Number):void
{
    // do something with parameters
}
```

- Pour créer un constructeur (une méthode spéciale appelée pendant la création d'une occurrence de classe), créez une méthode dont le nom correspond exactement au nom de la classe :

```
public function MyClass()
{
    // do stuff to set initial values for properties
    // and otherwise set up the object
    textVariable = "Hello there!";
    dateVariable = new Date(2001, 5, 11);
}
```

  Si vous n'incluez aucune méthode constructeur dans votre classe, le compilateur crée automatiquement un constructeur vide (sans paramètres ni instructions) dans votre classe.

Il est possible de définir d'autres éléments de classe, mais avec plus d'implication.

- Les *accesseurs* constituent un croisement spécial entre une méthode et une propriété. Lorsque vous écrivez le code de définition d'une classe, l'accesseur s'écrit comme une méthode, de manière à ce que vous puissiez accomplir plusieurs actions, et pas seulement la lecture ou l'attribution d'une valeur, seules actions disponibles lors de la définition d'une propriété. Toutefois, lorsque vous créez une occurrence de votre classe, vous traitez l'accesseur comme une propriété, en utilisant uniquement son nom pour lire ou attribuer la valeur. Pour plus d'informations, consultez la section « Méthodes accesseur get et set » à la page 104
- Dans ActionScript, les événements ne sont pas définis à l'aide d'une syntaxe spécifique. Pour définir les événements de votre classe, vous utilisez la fonctionnalité de la classe EventDispatcher qui permet de suivre les écouteurs d'événement et de notifier les événements qui surviennent. Pour plus d'informations sur la création d'événements dans vos propres classes,·consultez le chapitre « Gestion des événements » à la page 254.

Exemple : création d'une application de base

Il est possible de créer des fichiers source ActionScript externes, dotés de l'extension .as, dans plusieurs applications : Flash, Flex Builder, Dreamweaver ou tout éditeur de texte.

ActionScript 3.0 peut s'utiliser dans divers environnements de développement d'application, notamment l'outil de programmation Flash et Flex Builder.

Cette section étudie les différentes étapes de la création et de l'amélioration d'une simple application ActionScript 3.0 à l'aide de l'outil de programmation Flash ou de Flex Builder L'application à élaborer présente une manière simple d'utiliser les fichiers de classe externes ActionScript 3.0 dans Flash et Flex. Cette méthode peut s'appliquer à tous les autres exemples d'application proposés dans ce manuel.

Conception d'une application ActionScript

Vous devez avoir une idée de l'application à élaborer avant de commencer.

La représentation de cette conception peut être très simple (le nom de l'application et une brève description de son utilité) ou très compliquée (un ensemble de cahiers des charges contenant de nombreux diagrammes UML). Le but de ce manuel n'est pas de discuter en détail des principes de conception d'un logiciel. Il est toutefois important de garder à l'esprit que la conception de l'application est une étape essentielle du développement dans ActionScript.

Notre premier exemple d'application ActionScript est une application standard de salutation, du type « Hello World». Sa conception est donc très simple:

- L'application se nommera HelloWorld.
- Elle affichera un seul champ de texte contenant les mots « Hello World! ».
- Pour qu'elle soit facilement réutilisable, elle utilisera une seule classe orientée objet, appelée Greeter, qui pourra être exploitée dans un document Flash ou une application Flex.
- Une fois la version de base créée, vous ajouterez d'autres fonctionnalités, pour inviter l'utilisateur à saisir son nom et comparer ce nom à une liste d'utilisateurs reconnus.

Cette définition bien établie, vous pouvez commencer à créer l'application elle-même.

Création du projet HelloWorld et de la classe Greeter

Selon les décisions de conception, le code de l'application HelloWorld doit être facilement réutilisable. Pour satisfaire à cette exigence, l'application utilise une seule classe orientée objet, appelée Greeter, qui est exploitée au sein de l'application que vous allez créer dans Flex Builder ou l'outil de programmation Flash.

Pour créer la classe Greeter dans l'outil de programmation Flash :

1. Dans l'outil de programmation Flash, choisissez Fichier > Nouveau.
2. Dans la boîte de dialogue Nouveau document, sélectionnez un fichier ActionScript et cliquez sur OK.

   Une nouvelle fenêtre de modification ActionScript s'affiche.
3. Choisissez Fichier > Enregistrer. Sélectionnez un dossier pour votre application, nommez le fichier ActionScript **Greeter.as**, puis cliquez sur OK.

   Passez maintenant à la section « Ajout de code à la classe Greeter » à la page 32.

Ajout de code à la classe Greeter

La classe Greeter définit un objet, `Greeter`, que vous pourrez utiliser dans votre application HelloWorld.

Pour ajouter du code à la classe Greeter :

1 Tapez le code suivant dans le nouveau fichier :

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

La classe Greeter comporte une méthode unique, `sayHello()`, qui renvoie la chaîne « Hello World! ».

2 Sélectionnez Fichier > Enregistrer pour enregistrer ce fichier ActionScript.

La classe Greeter peut maintenant être utilisée dans une application.

Création d'une application utilisant votre code ActionScript

La classe Greeter que vous avez créée définit un ensemble autonome de fonctions logicielles, mais ne constitue pas pour autant une application complète. Pour l'utiliser, vous devez créer un document Flash ou une application Flex.

L'application HelloWorld crée une nouvelle occurrence de la classe Greeter. La procédure d'association de la classe Greeter à votre application est la suivante.

Pour créer une application ActionScript à l'aide de l'outil de programmation Flash :

1 Choisissez Fichier > Nouveau.

2 Dans la boîte de dialogue Nouveau document, sélectionnez Document Flash et cliquez sur OK.

Une nouvelle fenêtre Flash apparaît.

3 Choisissez Fichier > Enregistrer. Sélectionnez le même dossier qui contient le fichier de classe Greeter.as, nommez le document Flash **HelloWorld.fla**, puis cliquez sur OK.

4 Dans la palette des outils de Flash, sélectionnez l'outil Texte et faites-le glisser sur la scène pour définir un nouveau champ de texte d'environ 300 pixels de largeur et 100 pixels de hauteur.

5 Le champ de texte étant sélectionné sur la scène, dans le panneau Propriétés, définissez le type de texte sur Texte dynamique et tapez **mainText** comme nom d'occurrence du champ de texte.

6 Cliquez sur la première image du scénario principal.

7 Dans le panneau Actions, tapez le script suivant :

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

8 Enregistrez le fichier.

Passez maintenant à la section « Publication et test de votre application ActionScript » à la page 33.

Publication et test de votre application ActionScript

Le développement logiciel est un processus itératif. Vous écrivez du code, essayez de le compiler, puis modifiez ce code jusqu'à ce que la compilation soit réussie. Vous exécutez, puis testez l'application compilée, pour voir si elle répond aux intentions de conception. Si ce n'est pas le cas, vous modifiez à nouveau le code, jusqu'à obtenir satisfaction. Les environnements de développement Flash et Flex Builder offrent plusieurs manières de publier, tester et déboguer vos applications.

Voici une liste de base des étapes de test de l'application HelloWorld dans chacun de ces environnements.

Pour publier et tester une application ActionScript à l'aide de l'outil de programmation Flash :

1 Publiez votre application et recherchez les erreurs de compilation. Dans l'outil de programmation Flash, sélectionnez Contrôle > Tester l'animation pour compiler votre code ActionScript et exécutez l'application HelloWorld.

2 Si des erreurs ou des avertissements s'affichent dans la fenêtre Sortie lorsque vous testez votre application, résolvez-les dans les fichiers HelloWorld.fla ou HelloWorld.as, puis essayez de nouveau de tester l'application.

3 S'il n'existe aucune erreur de compilation, une fenêtre Flash Player affiche l'application HelloWorld.

Vous venez de créer une application orientée objet, simple mais complète, qui utilise ActionScript 3.0. Passez maintenant à la section « Amélioration de l'application HelloWorld » à la page 33.

Amélioration de l'application HelloWorld

Pour rendre l'application un peu plus attrayante, vous allez maintenant faire en sorte qu'elle demande le nom de l'utilisateur et le compare à une liste prédéfinie de noms.

Tout d'abord, vous devez mettre à jour la classe Greeter de manière à ajouter cette fonctionnalité. Vous devez ensuite mettre à jour l'application afin d'exploiter cette fonctionnalité.

Pour mettre à jour le fichier Greeter.as :

1 Ouvrez le fichier Greeter.as.

2 Remplacez son contenu par le suivant (les lignes nouvelles et modifiées sont signalées en gras) :

```
package
{
    public class Greeter
    {
        /**
         * Defines the names that should receive a proper greeting.
         */
        public static var validNames:Array = ["Sammy", "Frank", "Dean"];

        /**
         * Builds a greeting string using the given name.
         */
        public function sayHello(userName:String = ""):String
        {
            var greeting:String;
            if (userName == "")
            {
                greeting = "Hello. Please type your user name, and then press
                            the Enter key.";
            }
            else if (validName(userName))
```

```
            {
                greeting = "Hello, " + userName + ".";
            }
            else
            {
                greeting = "Sorry " + userName + ", you are not on the list.";
            }
            return greeting;
        }

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

La classe Greeter présente maintenant plusieurs fonctions nouvelles :

- Le tableau `validNames` répertorie les noms d'utilisateur valables. Lors du chargement de la classe Greeter, ce tableau contient trois noms.
- La méthode `sayHello()` accepte désormais un nom d'utilisateur et modifie la salutation en fonction de certaines conditions. Si le nom d'utilisateur `userName` est une chaîne vide (`""`), la propriété `greeting` permet de demander le nom de l'utilisateur. Si le nom d'utilisateur est valable, la salutation devient `"Hello, `*userName*`."` Enfin, si l'une de ces deux conditions n'est pas satisfaite, la variable `greeting` renvoie la valeur `"Sorry, `*userName*`, you are not on the list."` (Désolé, nom d'utilisateur, vous n'êtes pas sur la liste).
- La méthode `validName()` renvoie la valeur `true` si le nom entré `inputName` figure dans le tableau `validNames`, et la valeur `false` s'il ne s'y trouve pas. L'instruction `validNames.indexOf(inputName)` compare toutes les chaînes du tableau `validNames` à la chaîne `inputName`. La méthode `Array.indexOf()` renvoie la position d'index de la première occurrence d'un objet dans un tableau ou la valeur -1 si l'objet ne s'y trouve pas.

Ensuite, vous devez modifier le fichier Flash ou Flex qui référence cette classe ActionScript.

Pour modifier l'application à l'aide de l'outil de programmation Flash :

1. Ouvrez le fichier HelloWorld.fla.
2. Modifiez le script dans l'image 1 de façon à ce qu'une chaîne vide (`""`) soit transmise à la méthode `sayHello()` de la classe Greeter:

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

3. Sélectionnez l'outil Texte dans la palette des outils, puis créez deux champs de texte sur la scène, l'un à côté de l'autre, et directement sous le champ de texte `mainText` existant.
4. Dans le premier nouveau champ de texte, tapez le texte **User Name:** qui servira d'étiquette.

5 Sélectionnez l'autre nouveau champ de texte et dans l'Inspecteur des Propriétés, sélectionnez InputText comme type de champ de texte. Sélectionnez le type de ligne Une seule ligne. Tapez **textIn** comme nom d'occurrence.

6 Cliquez sur la première image du scénario principal.

7 Dans le panneau Actions, ajoutez les lignes suivantes à la fin du script existant :

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

textIn.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);

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

Le nouveau code ajoute la fonctionnalité suivante :

- Les deux premières lignes définissent les bordures de deux champs de texte.
- Un champ de texte d'entrée tel que le champ `textIn` a un ensemble d'événements qu'il peut envoyer. La méthode `addEventListener()` vous permet de définir une fonction exécutée lorsqu'un type d'événement se produit. Dans ce cas, cet événement est le fait d'appuyer sur une touche du clavier.
- La fonction personnalisée `keyPressed()` vérifie si la touche actionnée est la touche Entrée. Si tel est le cas, elle appelle la méthode `sayHello()` de l'objet `myGreeter`, en transmettant le texte du champ de texte `textIn` comme paramètre. Cette méthode renvoie une chaîne greeting en fonction de la valeur transmise. La chaîne renvoyée est ensuite affectée à la propriété `text` du champ de texte `mainText`.

Le script complet pour l'image 1 est le suivant :

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

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

textIn.addEventListener(KeyboardEvent.KEY_DOWN, keyPressed);

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

8 Enregistrez le fichier.

9 Choisissez Contrôle > Tester l'animation pour exécuter l'application.

Lorsque vous exécutez l'application, il vous est demandé d'entrer un nom d'utilisateur. S'il est valide (Sammy, Frank, ou Dean), l'application affiche le message de confirmation « hello ».

Exécution des exemples suivants

Maintenant que vous avez développé et exécuté l'application ActionScript 3.0 HelloWorld, vous devez disposer des connaissances suffisantes pour exécuter les autres exemples de code proposés dans ce manuel.

Test des exemples de code contenus dans un chapitre

Au fur et à mesure que vous avancez dans ce manuel, vous pouvez tester les exemples de code utilisés pour illustrer les différentes rubriques. Ce test peut impliquer l'affichage de la valeur des variables à certains stades du programme, ou bien l'affichage ou l'interaction avec un contenu à l'écran. Pour tester le contenu visuel ou l'interaction, les éléments nécessaires seront décrits avant ou dans le code ; il vous suffira de créer un document avec les éléments comme indiqué pour tester le code. Si vous souhaitez afficher la valeur d'une variable à un certain stade du programme, vous disposez de différentes façons pour le faire. Vous pouvez utiliser un débogueur tel que ceux intégrés dans Flex Builder et Flash. Pour un test simple, néanmoins, il peut être plus facile d'imprimer les valeurs des variables pour les visualiser.

Les étapes suivantes vous aideront à créer un document Flash que vous pouvez utiliser pour tester un code et afficher des valeurs de variables :

Pour créer un document Flash afin de tester des exemples contenus dans un chapitre :

1. Créez un document Flash et enregistrez-le sur votre disque dur.
2. Pour afficher des valeurs de test dans un champ de texte sur la scène, activez l'outil Texte et créez un champ de texte dynamique sur la scène. Un champ de texte haut et large avec le type de ligne défini sur Multiligne et la bordure activée sera très utile. Dans l'Inspecteur des Propriétés, donnez un nom d'occurrence au champ de texte (par exemple, outputText). Pour écrire des valeurs dans le champ de texte, vous ajouterez un code qui appelle la méthode `appendText()` sur l'exemple de code (décrit ci-dessous).
3. Vous pouvez également ajouter un appel de la fonction `trace()` au code (comme décrit ci-dessous) pour afficher les résultats de l'exemple.
4. Pour tester un exemple donné, copiez le code dans le panneau Actions`; si nécessaire, ajoutez un appel de la fonction` trace() `ou ajoutez une valeur au champ de texte à l'aide de sa méthode` appendText().
5. Dans le menu principal, choisissez Contrôle > Tester l'animation pour créer un fichier SWF et afficher les résultats.

Comme cette technique permet d'afficher les valeurs des variables, vous disposez de deux méthodes simples d'affichage des valeurs des variables lorsque vous testez les exemples : vous pouvez écrire les valeurs dans une occurrence de champ de texte sur la scène ou utilisez la fonction `trace()` pour imprimer les valeurs dans le panneau Sortie.

- Fonction `trace()` : la fonction `trace()` d'ActionScript écrit les valeurs des paramètres qui lui sont transmis (variables ou expressions littérales) sur le panneau Sortie. Un grand nombre des exemples fournis dans ce manuel comprend déjà un appel de la fonction `trace()`. Par conséquent, pour ces exemples, il vous suffit de copier le code dans votre document et de tester le projet. Si vous souhaitez utiliser `trace()` pour tester la valeur d'une variable dans un code qui ne la contient pas encore, ajoutez un appel `trace()` au code en transmettant la variable comme paramètre. Par exemple, si vous avez rencontré un code tel que celui-ci dans le chapitre,

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

  vous pouvez copier le code dans le panneau Actions, ajoutez ensuite un appel à la fonction `trace()` tel que celui-ci pour tester le résultat du code :

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

  Lorsque vous exécutez le programme, la ligne suivante est imprimée :

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

Chaque appel de la fonction `trace()` peut prendre plusieurs paramètres, qui se suivent tous et forment une unité représentée par une ligne imprimée. Un saut de ligne est ajouté à la fin de chaque appel de la fonction `trace()`. Par conséquent, des appels `trace()` séparés sont imprimés sur des lignes séparées.

- Champ de texte sur la scène : si vous préférez ne pas utiliser la fonction `trace()`, vous pouvez ajouter un champ de texte dynamique sur la scène à l'aide de l'outil Texte et écrire les valeurs dans ce champ de texte de façon à afficher les résultats d'un code. La méthode `appendText()` de la classe TextField permet d'ajouter une valeur String à la fin du contenu du champ de texte. Pour accéder au champ de texte à l'aide d'ActionScript, vous devez lui donner un nom d'occurrence dans l'Inspecteur des Propriétés. Par exemple, si votre champ de texte a le nom d'occurrence `outputText`, le code suivant permet de vérifier la valeur de la variable `albumName` :

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

Ce code écrit le texte suivant dans le champ de texte `outputText` :

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

Comme le montre l'exemple, la méthode `appendText()` ajoutera le texte sur la même ligne que le contenu précédent. Par conséquent, il est possible d'ajouter plusieurs valeurs sur la même ligne de texte à l'aide de plusieurs appels `appendText()`. Pour que le texte continue sur la ligne suivante, vous pouvez ajouter un caractère de nouvelle ligne (`"\n"`) :

```
outputText.appendText("\n"); // adds a line break to the text field
```

Contrairement à la fonction `trace()`, la méthode `appendText()` accepte une seule valeur comme paramètre. Cette valeur doit être une chaîne (une occurrence de String ou un littéral de chaîne). Pour imprimer la valeur d'une variable qui n'est pas une chaîne, vous devez d'abord convertir la valeur en une chaîne. Pour cela, la meilleure façon est d'appeler la méthode `toString()` de l'objet :

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

Utilisation des exemples de fin de chapitre

A l'instar de ce chapitre, la plupart des chapitres de ce manuel contiennent un exemple de fin de chapitre qui reprend un grand nombre des concepts traités. Néanmoins, contrairement à l'exemple Hello World de ce chapitre, ces exemples ne seront pas présentés avec des explications détaillées. Dans chaque exemple, le code ActionScript 3.0 considéré sera mis en évidence et étudié, mais aucune instruction ne sera fournie sur son exécution dans un environnement de développement spécifique. Néanmoins, les fichiers exemple fournis avec ce manuel incluent tous les fichiers nécessaires à la compilation et à l'exécution des exemples, selon l'environnement que vous aurez choisi.

Chapitre 4 : Syntaxe et langage ActionScript

ActionScript 3.0 comprend le langage ActionScript de base et l'interface de programmation d'applications d'Adobe Flash Player (API). Le langage de base correspond à la partie d'ActionScript qui définit la syntaxe du langage, ainsi que les types de données de plus haut niveau. ActionScript 3.0 fournit un accès à Flash Player par programmation.

Ce chapitre propose une brève introduction à la syntaxe et au langage ActionScript de base. Il explique comment utiliser des types de données, des variables et la syntaxe appropriée et comment contrôler le flux de données entrant dans votre programme.

Présentation du langage

Les objets sont au centre du langage ActionScript 3.0 : ils constituent ses éléments fondamentaux. Chaque variable que vous déclarez, chaque fonction que vous écrivez et chaque occurrence de classe que vous créez est un objet. Un programme ActionScript 3.0 peut être comparé à un groupe d'objets qui effectuent des tâches, répondent à des événements et communiquent entre eux.

Les programmeurs qui connaissent la programmation orientée objets en langage Java ou C++ assimilent peut-être les objets à des modules contenant deux sortes de membres : les données stockées dans des variables ou des propriétés de membres et le comportement accessible par le biais de méthodes. ActionScript 3.0 définit les objets de manière similaire, mais légèrement différente. Dans ActionScript 3.0, les objets ne sont que des collections de propriétés. Ces propriétés sont des conteneurs pouvant contenir non seulement des données, mais également des fonctions ou d'autres objets. Si une fonction est associée à un objet de cette façon, il s'agit d'une méthode.

Alors qu'en théorie, la définition d'ActionScript 3.0 peut paraître un peu étrange aux programmeurs ayant l'habitude d'utiliser le langage Java ou C++, en pratique, la définition de types d'objets avec des classes ActionScript 3.0 est très semblable à la façon dont les classes sont définies dans Java ou C++. Il est important de distinguer les deux définitions d'objet lors de la présentation du modèle d'objet ActionScript et d'autres rubriques plus techniques, mais généralement, le terme *propriétés* fait référence à des variables de membre de classe, par opposition aux méthodes. Dans le Guide de référence du langage et des composants ActionScript 3.0, par exemple, le terme *propriétés* signifie variables ou propriétés d'instruments de lecture et de définition. Il utilise le terme *méthodes* pour des fonctions faisant partie d'une classe.

Une légère différence entre les classes dans ActionScript et les classes dans Java ou C++ réside dans le fait que dans ActionScript, elles ne sont pas que des entités abstraites. Les classes ActionScript sont représentées par des *objets de classe* qui stockent les méthodes et les propriétés de la classe. Ainsi, des techniques pouvant sembler étranges pour les programmeurs de Java et C++ comme l'insertion d'instructions ou de code exécutable au niveau supérieur d'une classe ou d'un package sont autorisées.

Une autre différence entre les classes ActionScript et les classes Java ou C++ réside dans le fait que chaque classe ActionScript possède un *objet prototype*. Dans les versions précédentes d'ActionScript, les objets prototypes, liés les uns aux autres en *chaînes de prototypes,* servaient de base à l'ensemble de la hiérarchie d'héritage de classe. Dans ActionScript 3.0, néanmoins, les objets prototypes ne jouent qu'un rôle mineur dans le système d'héritage. Cependant, l'objet prototype peut toujours être utile comme solution de rechange aux méthodes et aux propriétés statiques si vous souhaitez partager une propriété et sa valeur parmi toutes les occurrences d'une classe.

Auparavant, les programmeurs ActionScript expérimentés pouvaient manipuler directement la chaîne de prototypes avec des éléments de langage intégrés spéciaux. Maintenant que le langage fournit une implémentation plus avancée d'une interface de programmation basée sur des classes, un grand nombre de ces éléments de langage spéciaux (`__proto__` et `__resolve,`, par exemple) ne font plus partie du langage. De plus, les optimisations du mécanisme d'héritage interne qui améliorent nettement les performances de Flash Player et Adobe AIR empêchent d'accéder directement au mécanisme.

Objets et classes

Dans ActionScript 3.0, chaque objet est défini par une classe. Une classe peut être considérée comme un modèle pour un type d'objet. Les définitions de classe peuvent inclure des variables et des constantes, qui comprennent des valeurs de données, et des méthodes, qui sont des fonctions encapsulant le comportement lié à la classe. Les valeurs stockées dans les propriétés peuvent être des *valeurs primitives* ou d'autres objets. Les valeurs primitives sont des nombres, des chaînes ou des valeurs booléennes.

ActionScript contient de nombreuses classes intégrées faisant partie du langage de base. Certaines de ces classes intégrées (Number, Boolean et String, par exemple), représentent les valeurs primitives disponibles dans ActionScript. D'autres, telles que les classe Array, Math et XML, définissent des objets plus complexes.

Toutes les classes, qu'elles soient intégrées ou définies par l'utilisateur, dérivent de la classe Object. Pour les programmeurs ayant une expérience avec ActionScript, il est important de noter que le type de données Object n'est plus le type de données par défaut, même si toutes les autres classes en dérivent. Dans ActionScript 2.0, les deux lignes de code suivantes étaient équivalentes car l'absence d'une annotation de type signifiait qu'une variable aurait été de type Object :

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

ActionScript 3.0, néanmoins, présente le concept de variables non typées, qui peuvent être désignées des deux façons suivantes :

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

Une variable non typée est différente d'une variable de type Object. La différence majeure est que les variables non typées peuvent contenir la valeur spéciale `undefined`, alors qu'une variable de type Object ne le peut pas.

Vous pouvez définir vos propres classes à l'aide du mot-clé `class`. Vous disposez de trois façons différentes pour déclarer les propriétés d'une classe : vous pouvez définir des constantes avec le mot-clé `const`, des variables avec le mot-clé `var` et des propriétés de lecture et de définition au moyen des attributs `get` et `set` dans une déclaration de méthode. Vous pouvez déclarer des méthodes avec le mot-clé `function`.

Vous créez une occurrence d'une classe à l'aide de l'opérateur `new`. L'exemple suivant crée une occurrence de la classe Date appelée `myBirthday`.

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

Packages et espaces de noms

Les packages et les espaces de noms sont des concepts associés. Les packages vous permettent de regrouper des définitions de classe de façon à faciliter le partage de code et à réduire les conflits de noms. Les espaces de noms vous permettent de contrôler la visibilité des identifiants (noms de méthode et de propriété, par exemple) et peuvent être appliqués à un code se trouvant à l'intérieur ou à l'extérieur d'un package. Utilisez des packages pour organiser vos fichiers de classe et des espaces de noms pour gérer la visibilité des méthodes et des propriétés individuelles.

Packages

Les packages dans ActionScript 3.0 sont implémentés avec des espaces de noms, mais ils ne sont pas synonymes. Lorsque vous déclarez un package, vous créez implicitement un type d'espace de noms spécial qui est sûr d'être connu lors de la compilation. Les espaces de noms, lorsque vous les créez explicitement, ne sont pas nécessairement connus au moment de la compilation.

L'exemple suivant utilise la directive `package` pour créer un simple package contenant une classe :

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

Le nom de la classe dans cet exemple est SampleCode. Etant donné que la classe se trouve à l'intérieur du package samples, le compilateur qualifie automatiquement le nom de classe lors de la compilation sous la forme de son nom qualifié : samples.SampleCode. Le compilateur qualifie également les noms des propriétés ou des méthodes, de sorte que `sampleGreeting` et `sampleFunction()` deviennent respectivement `samples.SampleCode.sampleGreeting` et `samples.SampleCode.sampleFunction()`.

Un grand nombre de développeurs (notamment ceux ayant une expérience de programmation Java) placent uniquement des classes au niveau supérieur d'un package. ActionScript 3.0, cependant, prend non seulement en charge des classes au niveau supérieur d'un package, mais également des variables, des fonctions et même des instructions. Une utilisation avancée de cette fonction consiste à définir un espace de noms au niveau supérieur d'un package de façon à ce que toutes les classes contenues dans ce dernier puissent l'utiliser. Il convient néanmoins de noter que seuls deux spécificateurs d'accès, `public` et `internal`, sont autorisés au niveau supérieur d'un package. Contrairement à Java qui vous permet de déclarer des classes imbriquées privées, ActionScript 3.0 ne prend en charge ni les classes imbriquées, ni les classes privées.

Néanmoins, les packages ActionScript 3.0 et les packages du langage de programmation Java présentent d'autres ressemblances. Comme vous pouvez l'observer dans l'exemple précédent, les références de package entièrement qualifiées sont exprimées à l'aide de l'opérateur point (.), comme dans Java. Vous pouvez utiliser des packages pour organiser votre code dans une structure hiérarchique intuitive que d'autres programmeurs peuvent utiliser. Ceci facilite le partage de code en vous permettant de créer votre propre package et de le partager avec d'autres personnes, et d'utiliser des packages créés par d'autres personnes dans votre code.

L'utilisation de packages vous permet également de vérifier que les noms d'identifiant que vous utilisez sont uniques et qu'ils ne sont pas incompatibles avec d'autres noms d'identifiant. Pour certaines personnes, il s'agit d'ailleurs de l'avantage principal des packages. Par exemple, deux programmeurs qui souhaitent partager leur code peuvent chacun avoir créé une classe appelée SampleCode. Sans packages, il y aurait un conflit de noms, et la seule façon de résoudre le problème serait de renommer l'une des classes. Mais avec des packages, vous pouvez éviter le conflit de noms en plaçant l'une des classes (ou de préférence les deux) dans des packages avec des noms uniques.

Vous pouvez également inclure des points intégrés dans le nom de votre package afin de créer des packages imbriqués. Ceci vous permet de créer une organisation hiérarchique des packages. Le package flash.xml, fourni par ActionScript 3.0, en est un bon exemple. Il est imbriqué au sein du package flash.

Le package flash.xml contient le programme d'analyse XML hérité utilisé dans les versions précédentes d'ActionScript. Il se trouve maintenant dans le package flash.xml car le nom de la classe XML héritée est incompatible avec celui de la nouvelle classe XML qui implémente le code XML pour la fonctionnalité de la spécification ECMAScript (E4X) disponible dans ActionScript 3.0.

Même s'il est judicieux de commencer par déplacer la classe XML héritée dans un package, la plupart des utilisateurs des classes XML héritées importent le package flash.xml, ce qui provoque le même conflit de noms à moins que vous utilisiez toujours le nom entièrement qualifié de la classe XML héritée (flash.xml.XML). Pour parer à ce problème, la classe XML héritée est maintenant appelée XMLDocument, comme l'indique l'exemple suivant :

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

La majeure partie d'ActionScript 3.0 se trouve dans le package flash. Par exemple, le package flash.display contient l'API de la liste d'affichage, et le package flash.events contient le nouveau modèle d'événement.

Création de packages

ActionScript 3.0 vous permet d'organiser vos packages, classes et fichiers source avec une grande souplesse. Les versions précédentes d'ActionScript autorisaient uniquement une classe par fichier source et exigeaient que le nom du fichier source corresponde au nom de la classe. ActionScript 3.0 vous permet d'inclure plusieurs classes dans un fichier source, mais une seule classe dans chaque fichier peut être utilisée par le code externe à ce fichier. En d'autres termes, une seule classe dans chaque fichier peut être déclarée à l'intérieur d'une déclaration de package. Vous devez déclarer toute classe supplémentaire à l'extérieur de votre définition de package. Elles sont alors invisibles pour le code externe à ce fichier source. Le nom de la classe déclarée à l'intérieur de la définition de package doit correspondre au nom du fichier source.

ActionScript 3.0 permet également de déclarer des packages avec davantage de flexibilité. Dans les versions précédentes d'ActionScript, les packages représentaient simplement des répertoires dans lesquels vous placiez des fichiers source, et vous ne déclariez pas les packages avec l'instruction `package` mais incluiez plutôt le nom du package dans le nom de classe complet dans votre déclaration de classe. Même si les packages continuent à représenter des répertoires dans ActionScript 3.0, leur contenu n'est pas limité aux seules classes. Dans ActionScript 3.0, vous utilisez l'instruction `package` pour déclarer un package, ce qui signifie que vous pouvez également déclarer des variables, des fonctions et des espaces de noms au niveau supérieur d'un package. Vous pouvez également y inclure des instructions exécutables. Si vous déclarez des variables, des fonctions ou des espaces de noms à ce niveau, les seuls attributs disponibles sont `public` et `internal`, et une seule déclaration au niveau du package par fichier peut utiliser l'attribut `public`, que cette déclaration soit une classe, une variable, une fonction ou un espace de nom.

Les packages sont utiles pour organiser votre code et éviter les conflits de noms. Vous ne devez pas confondre le concept de packages avec le concept distinct d'héritage de classe. Deux classes se trouvant dans le même package ont un espace de noms en commun mais elles ne sont pas nécessairement liées l'une à l'autre. De même, il se peut qu'un package imbriqué n'ait aucun lien sémantique avec son package parent.

Importation de packages

Si vous souhaitez utiliser une classe se trouvant à l'intérieur d'un package, vous devez importer soit le package soit la classe en question. Ceci diffère d'ActionScript 2.0 où l'importation de classes était facultative.

Par exemple, revenons à l'exemple de classe SampleCode fourni précédemment dans ce chapitre. Si la classe se trouve dans un package appelé samples, vous devez utiliser l'une des instructions d'importation suivantes avant d'utiliser la classe SampleCode :

```
import samples.*;
```

ou

```
import samples.SampleCode;
```

En général, les instructions `import` doivent être aussi spécifiques que possible. Si vous envisagez d'utiliser uniquement la classe SampleCode issue du package samples, importez uniquement la classe SampleCode au lieu du package entier auquel elle appartient. L'importation des packages entiers peut provoquer des conflits de noms inattendus.

Vous devez également placer le code source qui définit le package ou la classe dans votre *chemin de classe*. Le chemin de classe est une liste définie par l'utilisateur de chemins de répertoire locaux qui détermine l'endroit où le compilateur recherche des classes et des packages importés. Le chemin de classe est parfois appelé *chemin de création* ou *chemin source*.

Une fois que vous avez importé correctement la classe ou le package, vous pouvez utiliser le nom entièrement qualifié de la classe (samples.SampleCode) ou simplement le nom de la classe (SampleCode).

Les noms entièrement qualifiés sont utiles lorsque des classes, des méthodes ou des propriétés ayant le même nom génèrent un code ambigu mais ils peuvent être difficiles à gérer si vous les utilisez pour tous les identifiants. Par exemple, l'utilisation du nom entièrement qualifié génère un code détaillé lorsque vous instanciez une occurrence de classe SampleCode :

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

Plus les niveaux de packages imbriqués augmentent, moins votre code est lisible. Lorsque vous pensez que des identifiants ambigus ne sont pas un problème, vous pouvez rendre votre code plus lisible en utilisant des identifiants simples. Par exemple, l'instanciation d'une nouvelle occurrence de la classe SampleCode est beaucoup moins longue si vous utilisez uniquement l'identifiant de classe :

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

Si vous tentez d'utiliser des noms d'identifiant sans importer au préalable la classe ou le package approprié, il est impossible pour le compilateur de trouver les définitions de classe. D'un autre côté, si vous importez un package ou une classe, toute tentative de définition d'un nom qui provoque un conflit avec un nom importé génère une erreur.

Lors de la création d'un package, le spécificateur d'accès par défaut pour tous les membres de ce package est `internal`, ce qui signifie que, par défaut, les membres du package ne sont visibles que pour d'autres membres de ce package. Si vous souhaitez qu'une classe soit disponible pour un code externe au package, vous devez la déclarer `public`. Par exemple, le package suivant contient deux classes, SampleCode et CodeFormatter :

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

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

La classe SampleCode est visible en dehors du package car elle est déclarée comme classe `public`. La classe CodeFormatter, cependant, est visible uniquement dans le package samples. Si vous tentez d'accéder à la classe CodeFormatter en dehors du package samples, vous générez une erreur, comme l'indique l'exemple suivant :

```
import samples.SampleCode;
import samples.CodeFormatter;
var mySample:SampleCode = new SampleCode(); // okay, public class
var myFormatter:CodeFormatter = new CodeFormatter(); // error
```

Si vous souhaitez que les deux classes soient disponibles en dehors du package, vous devez les déclarer `public`. Vous ne pouvez pas appliquer l'attribut `public` à la déclaration de package.

Les noms entièrement qualifiés sont utiles pour résoudre les conflits de noms pouvant se produire lors de l'utilisation de packages. Un tel scénario peut se produire si vous importez deux packages qui définissent des classes ayant le même identifiant. Par exemple, considérons le package suivant, qui possède également une classe appelée SampleCode :

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

Si vous importez les deux classes, comme suit, vous avez un conflit de noms lorsque vous vous référez à la classe SampleCode :

```
import samples.SampleCode;
import langref.samples.SampleCode;
var mySample:SampleCode = new SampleCode(); // name conflict
```

Le compilateur n'a aucun moyen de savoir quelle classe SampleCode il doit utiliser. Pour résoudre ce conflit, vous devez utiliser le nom entièrement qualifié de chaque classe, comme suit :

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

Remarque : *les programmeurs ayant une expérience C++ confondent souvent l'instruction `import` avec l'instruction `#include`. La directive `#include` est nécessaire dans C++ car les compilateurs C++ traitent un fichier à la fois et ne cherchent pas de définitions de classe dans d'autres fichiers, à moins qu'un fichier d'en-tête soit inclus explicitement. ActionScript 3.0 a une directive `include`, mais elle n'est pas conçue pour importer des classes et des packages. Pour importer des classes ou des packages dans ActionScript 3.0, vous devez utiliser l'instruction `import` et placer le fichier source qui contient le package dans le chemin de classe.*

Espaces de noms

Les espaces de noms vous permettent de contrôler la visibilité des propriétés et des méthodes que vous créez. Considérez les spécificateurs de contrôle d'accès `public`, `private`, `protected` et `internal` comme des espaces de noms intégrés. Si ces spécificateurs de contrôle d'accès prédéfinis ne répondent pas à vos besoins, vous pouvez créer vos propres espaces de noms.

Si vous avez l'habitude d'utiliser des espaces de noms XML, cette section ne vous fournira pas de nouvelles informations, bien que la syntaxe et les détails de l'implémentation d'ActionScript soient légèrement différents de ceux d'XML. Si vous n'avez jamais travaillé avec des espaces de noms, le concept est simple, mais l'implémentation possède une terminologie particulière que vous devrez apprendre.

Pour comprendre comment fonctionnent les espaces de noms, il convient de savoir que le nom d'une propriété ou d'une méthode contient deux parties : un identifiant et un espace de nom. L'identifiant est ce que vous considérez généralement comme un nom. Par exemple, les identifiants dans la définition de classe suivante sont `sampleGreeting` et `sampleFunction()` :

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

Lorsque les définitions ne sont pas précédées d'un attribut d'espace de nom, leurs noms sont qualifiés par l'espace de noms `internal` par défaut, ce qui signifie qu'ils sont visibles uniquement aux appelants du même package. Si le compilateur est défini sur le mode strict, il génère un avertissement indiquant que l'espace de noms `internal` s'applique à tout identifiant sans attribut d'espace de nom. Pour vérifier qu'un identifiant est disponible partout, vous devez spécifiquement faire précéder le nom de l'identifiant de l'attribut `public`. Dans l'exemple de code précédent, `sampleGreeting` et `sampleFunction()` ont une valeur d'espace de noms d'`internal`.

Vous devez suivre trois étapes de base lorsque vous utilisez des espaces de noms. Premièrement, vous devez définir l'espace de noms à l'aide du mot-clé `namespace`. Par exemple, le code suivant définit l'espace de noms `version1` :

```
namespace version1;
```

Deuxièmement, vous appliquez votre espace de noms en l'utilisant à la place d'un spécificateur de contrôle d'accès dans une déclaration de méthode ou de propriété. L'exemple suivant place une fonction appelée `myFunction()` dans l'espace de noms `version1` :

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

Troisièmement, une fois que vous avez appliqué l'espace de nom, vous pouvez le référencer à l'aide de la directive `use` ou en qualifiant le nom d'un identifiant avec un espace de nom. L'exemple suivant fait référence à la fonction `myFunction()` à l'aide de la directive `use` :

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

Vous pouvez également utiliser un nom qualifié pour référencer la fonction `myFunction()`, comme l'indique l'exemple suivant :

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

Définition des espaces de noms

Les espaces de noms contiennent une valeur, l'URI (Uniform Resource Identifier), parfois appelée *nom d'espace de nom*. Un URI vous permet de vérifier que votre définition d'espace de noms est unique.

Vous créez un espace de noms en déclarant sa définition de deux façons différentes. Vous pouvez soit définir un espace de noms avec un URI explicite, comme vous définiriez un espace de noms XML, soit omettre l'URI. L'exemple suivant indique comment un espace de noms peut être défini à l'aide d'un URI :

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

L'URI sert de chaîne d'identification unique pour cet espace de nom. Si vous omettez l'URI, comme dans l'exemple suivant, le compilateur crée une chaîne d'identification interne unique à la place de l'URI. Vous n'avez pas accès à cette chaîne d'identification interne.

```
namespace flash_proxy;
```

Une fois que vous avez défini un espace de noms (avec ou sans URI), vous ne pouvez pas le redéfinir dans le même domaine. Une tentative de définition d'un espace de noms ayant été défini dans le même domaine génère une erreur du compilateur.

Si un espace de noms est défini dans un package ou une classe, il risque de ne pas être visible au code externe à ce package ou à cette classe, à moins que vous utilisiez le spécificateur de contrôle d'accès approprié. Par exemple, le code suivant indique l'espace de noms `flash_proxy` défini dans le package flash.utils. Dans l'exemple suivant, l'absence de spécificateur de contrôle d'accès signifie que l'espace de noms `flash_proxy` est visible uniquement au code dans le package flash.utils et non au code externe au package :

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

Le code suivant utilise l'attribut `public` pour rendre l'espace de noms `flash_proxy` visible au code externe au package :

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

Application d'espaces de noms

Appliquer un espace de noms signifie placer une définition dans un espace de nom. Les définitions que vous pouvez placer dans des espaces de noms peuvent être des fonctions, des variables et des constantes (vous ne pouvez pas placer une classe dans un espace de noms personnalisé).

Supposez, par exemple, qu'une fonction soit déclarée à l'aide de l'espace de noms de contrôle d'accès `public`. L'utilisation de l'attribut `public` dans une définition de fonction place la fonction dans l'espace de noms public et la rend visible à tout le code. Une fois que vous avez défini un espace de nom, vous pouvez l'utiliser de la même façon que vous utiliseriez l'attribut `public`, et la définition est disponible au code pouvant référencer votre espace de noms personnalisé. Par exemple, si vous définissez un espace de noms `example1`, vous pouvez ajouter une méthode appelée `myFunction()` à l'aide de `example1` comme attribut, tel que l'indique l'exemple suivant :

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

Déclarer la méthode `myFunction()` à l'aide de l'espace de noms `example1` comme attribut signifie que la méthode appartient à l'espace de noms `example1`.

Tenez compte des points suivants lorsque vous appliquez des espaces de noms :

- Vous pouvez appliquer un seul espace de noms à chaque déclaration.
- Il n'existe aucun moyen d'appliquer un attribut d'espace de noms à plusieurs définitions simultanément. En d'autres termes, si vous souhaitez appliquer votre espace de noms à dix fonctions différentes, vous devez ajouter votre espace de noms comme attribut à chacune des dix définitions de fonction.
- Si vous appliquez un espace de nom, vous ne pouvez pas spécifier un spécificateur de contrôle d'accès car les espaces de noms et les spécificateurs de contrôle d'accès s'excluent mutuellement. En d'autres termes, vous ne pouvez pas déclarer une fonction ou une propriété comme `public`, `private`, `protected` ou `internal` si vous appliquez votre espace de nom.

Référence d'espaces de noms

Il est inutile de référencer explicitement un espace de noms lorsque vous utilisez une méthode ou une propriété déclarée avec l'un des espaces de noms de contrôle d'accès (`public`, `private`, `protected` et `internal`, par exemple). En effet, l'accès à ces espaces de noms spéciaux dépend du contexte. Par exemple, les définitions placées dans l'espace de noms `private` sont automatiquement disponibles pour le code dans la même classe. Pour les espaces de noms que vous définissez, cependant, le contexte ne compte pas. Pour utiliser une méthode ou une propriété que vous avez placée dans un espace de noms personnalisé, vous devez référencer l'espace de nom.

Vous pouvez référencer des espaces de noms avec la directive `use namespace` ou qualifier le nom avec l'espace de noms à l'aide du ponctuateur de qualificatif de nom (`::`). Le fait de référencer un espace de noms avec la directive `use namespace` ouvre l'espace de nom, ce qui permet de l'appliquer à n'importe quel identifiant non qualifié. Par exemple, si vous avez défini l'espace de noms `example1`, vous pouvez accéder aux noms dans cet espace de noms en utilisant `use namespace example1`:

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

Vous pouvez ouvrir plusieurs espaces de noms simultanément. Une fois que vous avez ouvert un espace de noms avec `use namespace`, il reste ouvert dans le bloc de code dans lequel il a été ouvert. Il n'existe aucun moyen pour fermer explicitement un espace de nom.

Néanmoins, le fait d'avoir plusieurs espaces de noms ouverts augmente la probabilité que des conflits de noms se produisent. Si vous préférez ne pas ouvrir d'espace de nom, vous pouvez éviter la directive `use namespace` en qualifiant le nom de la propriété ou de la méthode avec l'espace de noms et le ponctuateur de qualificatif de nom. Par exemple, le code suivant indique comment vous pouvez qualifier le nom `myFunction()` avec l'espace de noms `example1`:

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

Utilisation d'espaces de noms

Vous pouvez trouver un exemple d'espace de nom, tiré du monde réel, utilisé pour éviter des conflits sur les noms dans la classe flash.utils.Proxy qui fait partie d'ActionScript 3.0. La classe Proxy, qui remplace la propriété `Object.__resolve` d'ActionScript 2.0, vous permet d'intercepter les références aux propriétés ou aux méthodes non définies avant qu'une erreur ne se produise. Toutes les méthodes de la classe Proxy se trouvent dans l'espace de noms `flash_proxy` afin d'empêcher les conflits de noms.

Pour mieux comprendre comment l'espace de noms `flash_proxy` est utilisé, vous devez savoir comment utiliser la classe Proxy. La fonctionnalité de la classe Proxy est disponible uniquement aux classes qui héritent d'elle. En d'autres termes, si vous souhaitez utiliser les méthodes de la classe Proxy d'un objet, la définition de classe de l'objet doit étendre la classe Proxy. Par exemple, si vous souhaitez intercepter des tentatives d'appel d'une méthode non définie, vous devez étendre la classe Proxy puis remplacer la méthode `callProperty()` de la classe Proxy.

L'implémentation des espaces de noms est généralement un processus en trois étapes (définition, application et référence d'un espace de nom). Etant donné que vous n'appelez jamais explicitement une méthode de la classe Proxy, cependant, l'espace de noms `flash_proxy` est défini et appliqué uniquement, jamais référencé. ActionScript 3.0 définit l'espace de nom `flash_proxy` et l'applique dans la classe Proxy. Votre code doit uniquement appliquer l'espace de noms `flash_proxy` à des classes qui étendent la classe Proxy.

L'espace de noms `flash_proxy` est défini dans le package flash.utils comme illustré ci-dessous :

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

L'espace de noms est appliqué aux méthodes de la classe Proxy comme indiqué dans l'extrait suivant issu de la classe Proxy :

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

Comme l'indique le code suivant, vous devez d'abord importer la classe Proxy et l'espace de noms `flash_proxy`. Vous devez ensuite déclarer votre classe de façon à ce qu'elle étende la classe Proxy (vous devez également ajouter l'attribut `dynamic` si vous compilez en mode strict). Lorsque vous remplacez la méthode `callProperty()`, vous devez utiliser l'espace de noms `flash_proxy`.

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

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

Si vous créez une occurrence de la classe MyProxy et appelez une méthode non définie (la méthode `testing()` appelée dans l'exemple suivant, par exemple), votre objet Proxy intercepte l'appel de méthode et exécute les instructions se trouvant dans la méthode `callProperty()` remplacée (dans ce cas, une instruction `trace()` simple).

```
var mySample:MyProxy = new MyProxy();
mySample.testing(); // method call intercepted: testing
```

Le fait que les méthodes de la classe Proxy se trouvent dans l'espace de noms `flash_proxy` présente deux avantages. Premièrement, le fait d'avoir un espace de noms séparé réduit l'encombrement dans l'interface publique des classes qui étendent la classe Proxy. (Il existe environ douze méthodes dans la classe Proxy que vous pouvez remplacer. Elles ne sont pas conçues pour être appelées directement. Le fait de toutes les placer dans l'espace de noms public peut prêter à confusion.) Deuxièmement, le fait d'utiliser l'espace de noms `flash_proxy` évite les conflits de nom si votre sous-classe Proxy contient des méthodes d'occurrence avec des noms correspondant à l'une des méthodes de la classe Proxy. Par exemple, vous pouvez nommer l'une de vos méthodes `callProperty()`. Le code suivant est acceptable car votre version de la méthode `callProperty()` se trouve dans un espace de noms différent :

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

Vous pouvez également utiliser des espaces de noms pour accéder à des méthodes ou à des propriétés autrement qu'avec les quatre spécificateurs de contrôle d'accès (`public`, `private`, `internal` et `protected`). Par exemple, vous pouvez avoir des méthodes d'utilitaire éparpillées sur plusieurs packages. Vous souhaitez que ces méthodes soient disponibles pour tous vos packages, mais vous ne souhaitez pas qu'elles soient publiques. Pour cela, vous pouvez créer un espace de noms et l'utiliser comme spécificateur de contrôle d'accès spécial.

L'exemple suivant utilise un espace de noms défini par l'utilisateur pour regrouper deux fonctions se trouvant dans différents packages. En les regroupant dans le même espace de nom, vous pouvez rendre les deux fonctions visibles à une classe ou à un package au moyen d'une seule instruction `use namespace`.

Cet exemple utilise quatre fichiers pour démontrer la technique. Tous les fichiers doivent se trouver dans votre chemin de classe. Le premier fichier, myInternal.as, sert à définir l'espace de noms `myInternal`. Etant donné que le fichier se trouve dans un package appelé example, vous devez placer le fichier dans un dossier appelé example. L'espace de noms est marqué comme `public` pour pouvoir être importé dans d'autres packages.

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

Le deuxième et le troisième fichiers, Utility.as et Helper.as, définissent les classes qui contiennent des méthodes devant être disponibles pour d'autres packages. La classe Utility se trouve dans le package example.alpha, ce qui signifie que le fichier doit être placé dans un dossier appelé alpha qui est un sous-dossier du dossier example. La classe Helper se trouve dans le package example.beta, ce qui signifie que le fichier doit être placé dans un dossier appelé beta qui est également un sous-dossier du dossier example. Ces deux packages, example.alpha et example.beta, doivent importer l'espace de noms avant de l'utiliser.

```
// Utility.as in the example/alpha folder
package example.alpha
{
    import example.myInternal;

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

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

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

```
// Helper.as in the example/beta folder
package example.beta
{
    import example.myInternal;

    public class Helper
    {
        private static var _timeStamp:Date;

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

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

Le quatrième fichier, NamespaceUseCase.as, est la classe de l'application principale et doit être un frère pour le dossier example. Dans Adobe Flash CS4 Professional, cette classe ferait office de classe de document du fichier FLA. La classe NamespaceUseCase importe également l'espace de noms `myInternal` et l'utilise pour appeler les deux méthodes statiques qui résident dans les autres packages. L'exemple utilise des méthodes statiques pour simplifier le code uniquement. Les méthodes statiques et d'occurrence peuvent être placées dans l'espace de noms `myInternal`.

```
// NamespaceUseCase.as
package
{
    import flash.display.MovieClip;
    import example.myInternal; // import namespace
    import example.alpha.Utility;// import Utility class
    import example.beta.Helper;// import Helper class

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

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

            Helper.someTask();
            trace(Helper.lastCalled); // [time someTask() was last called]
        }
    }
}
```

Variables

Les variables vous permettent de stocker des valeurs que vous utilisez dans votre programme. Pour déclarer une variable, vous devez utiliser l'instruction `var` avec le nom de variable. Dans ActionScript 2.0, l'utilisation de l'instruction `var` est nécessaire uniquement si vous utilisez des annotations de type. Dans ActionScript 3.0, l'utilisation de l'instruction `var` est obligatoire. Par exemple, la ligne d'ActionScript suivante déclare une variable appelée `i` :

```
var i;
```

Si vous omettez l'instruction `var` lors de la déclaration d'une variable, vous obtenez une erreur de compilateur en mode strict et une erreur d'exécution en mode standard. Par exemple, la ligne de code suivante provoque une erreur si la variable `i` n'a pas été définie précédemment :

```
i; // error if i was not previously defined
```

Vous devez associer une variable à un type de données lorsque vous déclarez la variable. La déclaration d'une variable sans désigner son type est autorisée mais provoque un avertissement du compilateur en mode strict. Vous désignez le type d'une variable en ajoutant le nom de variable avec deux-points (:), suivi par le type de variable. Par exemple, le code suivant déclare une variable `i` de type int :

```
var i:int;
```

Vous pouvez affecter une valeur à une variable à l'aide de l'opérateur d'affectation (`=`). Par exemple, le code suivant déclare une variable `i` et lui affecte la valeur 20 :

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

Il peut être plus pratique d'affecter une valeur à une variable au moment où vous déclarez cette dernière, comme dans l'exemple suivant :

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

La technique d'affectation d'une valeur à une variable au moment de sa déclaration est couramment utilisée non seulement lors de l'affectation de valeurs primitives (nombres entiers et chaînes) mais également lors de la création d'un tableau ou de l'instanciation de l'occurrence d'une classe. L'exemple suivant illustre un tableau déclaré auquel est affectée une valeur à l'aide d'une ligne de code.

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

Vous pouvez créer une occurrence d'une classe à l'aide de l'opérateur `new`. L'exemple suivant crée une occurrence d'une classe appelée `CustomClass` et affecte une référence à l'occurrence de classe nouvellement créée sur la variable appelée `customItem` :

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

Si vous avez plusieurs variables à déclarer, vous pouvez le faire sur une ligne de code à l'aide de l'opérateur virgule (`,`) pour séparer les variables. Par exemple, le code suivant déclare trois variables sur une ligne de code :

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

Vous pouvez également affecter des valeurs à chacune des variables sur une même ligne de code. Par exemple, le code suivant déclare trois variables (`a`, `b`, et `c`) et affecte une valeur à chacune d'entre elles :

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

Bien que vous puissiez utiliser l'opérateur virgule pour regrouper des déclarations de variable dans une instruction, ceci risque de rendre votre code moins lisible.

Présentation du domaine d'une variable

Le *domaine* d'une variable est la partie de votre code dans laquelle une référence lexicale peut accéder à la variable. Une variable *globale* est une variable définie dans toutes les parties de votre code, alors qu'une variable *locale* est une variable définie dans une seule partie de votre code. Dans ActionScript 3.0, le domaine des variables est toujours limité à la fonction ou à la classe dans laquelle elles sont déclarées. Une variable globale est une variable que vous définissez en dehors de toute définition de classe ou de fonction. Par exemple, le code suivant crée une variable globale `strGlobal` en la déclarant en dehors de toute fonction. L'exemple indique qu'une variable globale est disponible à la fois à l'intérieur et à l'extérieur de la définition de fonction.

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

Vous déclarez une variable locale en déclarant la variable à l'intérieur d'une définition de fonction. La plus petite partie de code pour laquelle vous pouvez définir une variable locale est une définition de fonction. Une variable locale déclarée dans une fonction existe uniquement dans cette fonction. Par exemple, si vous déclarez une variable appelée `str2` dans une fonction appelée `localScope()`, cette variable ne sera pas disponible en dehors de cette fonction.

```
function localScope()
{
    var strLocal:String = "local";
}
localScope();
trace(strLocal); // error because strLocal is not defined globally
```

Si le nom utilisé pour votre variable locale est déjà déclaré en tant que variable globale, la définition locale masque la définition globale alors que la variable locale est dans le domaine. La variable globale persiste en dehors de la fonction. Par exemple, le code suivant crée une variable globale de type chaîne appelée `str1`, puis une variable locale du même nom dans la fonction `scopeTest()`. L'instruction `trace` placée dans la fonction génère la valeur locale de la variable, tandis que celle qui est placée en dehors de la fonction génère la valeur globale de la variable.

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

Les variables ActionScript, contrairement aux variables dans C++ et Java, n'ont pas de domaine de niveau bloc. Un bloc de code est un groupe d'instructions placé entre une accolade d'ouverture ({) et une accolade de fermeture (}). Dans certains langages de programmation tels que C++ et Java, les variables déclarées dans un bloc de code ne sont pas disponibles en dehors de ce bloc de code. Cette restriction de domaine est appelée domaine de niveau bloc et n'existe pas dans ActionScript. Si vous déclarez une variable à l'intérieur d'un bloc de code, elle est disponible non seulement dans ce bloc de code mais également dans d'autres parties de la fonction à laquelle le bloc de code appartient. Par exemple, la fonction suivante contient des variables définies dans différents domaines de bloc. Toutes les variables sont disponibles dans la fonction.

```
function blockTest (testArray:Array)
{
    var numElements:int = testArray.length;
    if (numElements > 0)
    {
        var elemStr:String = "Element #";
        for (var i:int = 0; i < numElements; i++)
        {
            var valueStr:String = i + ": " + testArray[i];
            trace(elemStr + valueStr);
        }
        trace(elemStr, valueStr, i); // all still defined
    }
    trace(elemStr, valueStr, i); // all defined if numElements > 0
}

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

Une implication intéressante de l'absence de domaine de niveau bloc est que vous pouvez lire ou écrire dans une variable avant de la déclarer, tant qu'elle est déclarée avant que la fonction se termine. Ceci est dû à une technique appelée *hissage*, qui signifie que le compilateur déplace toutes les déclarations de variable en haut de la fonction. Par exemple, le code suivant compile même si la fonction `trace()` initiale pour la variable `num` se produit avant la déclaration de la variable `num` :

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

Néanmoins, le compilateur ne hisse aucune instruction d'affectation. Ceci explique pourquoi la fonction `trace()` initiale de `num` fournit `NaN` (pas un nombre), qui est la valeur par défaut pour des variables du type de données Number. Cela signifie que vous pouvez affecter des valeurs à des variables même avant leur déclaration, comme indiqué dans l'exemple suivant :

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

Valeurs par défaut

Une *valeur par défaut* est le contenu d'une variable avant que vous définissiez sa valeur. Vous *initialisez* une variable lorsque vous lui affectez sa première valeur. Si vous déclarez une variable sans définir sa valeur, cette variable est *non initialisée*. La valeur d'une variable non initialisée dépend de son type de données. Le tableau suivant décrit les valeurs par défaut des variables, organisées par type de données :

Type de données	Valeur par défaut
Boolean	`false`
int	0
Number	`NaN`
Object	`null`
String	null

Type de données	Valeur par défaut
uint	0
Non déclaré (équivalent à l'annotation de type `*`)	`undefined`
Toutes les autres classes, y compris celles définies par l'utilisateur	`null`

Pour les variables de type Number, la valeur par défaut est `NaN` (pas un nombre), qui est une valeur spéciale définie par la norme IEEE-754 pour indiquer une valeur qui ne représente pas un nombre.

Si vous déclarez une variable sans déclarer son type de données, le type de données par défaut `*` s'applique, ce qui signifie que la variable n'est pas typée. Si vous n'initialisez pas non plus de variable non typée avec une valeur, sa valeur par défaut est `undefined`.

Pour les types de données autres que Boolean, Number, int, et uint, la valeur par défaut d'une variable non initialisée est `null`. Ceci s'applique à toutes les classes définies par ActionScript 3.0, ainsi qu'aux classes personnalisées que vous créez.

La valeur `null` n'est pas une valeur valide pour des variables de type Boolean, Number, int ou uint. Si vous tentez d'affecter la valeur `null` à une telle variable, la valeur est convertie en la valeur par défaut pour ce type de données. Pour les variables de type Object, vous pouvez affecter une valeur `null`. Si vous tentez d'affecter la valeur `undefined` à une variable de type Object, la valeur est convertie en `null`.

Pour les variables de type Number, il existe une fonction de niveau supérieur spéciale appelée `isNaN()` qui renvoie la valeur Boolean `true` si la variable n'est pas un nombre, et `false` autrement.

Types de données

Un *type de données* définit un ensemble de valeurs. Par exemple, le type de données Boolean est l'ensemble de deux valeurs exactement : `true` et `false`. Outre le type de données Boolean, ActionScript 3.0 définit plusieurs autres types de données couramment utilisés tels que String, Number et Array. Vous pouvez définir vos types de donnés en utilisant des classes ou des interfaces afin de définir un ensemble de valeurs personnalisé. Toutes les valeurs dans ActionScript 3.0, qu'elles soient primitives ou complexes, sont des objets.

Une *valeur primitive* est une valeur appartenant à l'un des types de données suivants : Boolean, int, Number, String et uint. L'utilisation de valeurs primitives est généralement plus rapide que l'utilisation de valeurs complexes car ActionScript stocke les valeurs primitives de façon à permettre des optimisations de vitesse et de mémoire.

Remarque : *pour les lecteurs intéressés par les détails techniques, ActionScript stocke les valeurs primitives en tant qu'objets inaltérables. Le fait qu'elles soient stockées en tant qu'objets inaltérables signifie que le transfert par référence est identique au transfert par valeur. Ceci réduit l'utilisation de la mémoire et augmente la vitesse d'exécution car les références sont généralement beaucoup plus petites que les valeurs elles-mêmes.*

Une *valeur complexe* est une valeur qui n'est pas une valeur primitive. Les types de données qui définissent des ensembles de valeurs complexes comprennent Array, Date, Error, Function, RegExp, XML et XMLList.

De nombreux langages de programmation font la distinction entre les valeurs primitives et leurs enveloppes. Java, par exemple, a une valeur primitive int et la classe java.lang.Integer qui l'enveloppe. Les valeurs primitives de Java ne sont pas des objets, mais leurs enveloppes le sont, ce qui rend les valeurs primitives utiles pour certaines opérations et les enveloppes pour d'autres. Dans ActionScript 3.0, les valeurs primitives et leurs enveloppes ne peuvent être distinguées, à des fins pratiques. Toutes les valeurs, même les valeurs primitives, sont des objets. Flash Player et Adobe AIR traitent ces types primitifs comme des cas spéciaux qui se comportent comme des objets n'exigeant pas le temps normal associé à la création d'objets. Cela signifie que les deux lignes de code suivantes sont équivalentes :

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

Tous les types de données primitifs et complexes répertoriés ci-dessus sont définis par les classes de base d'ActionScript 3.0. Les classes de base vous permettent de créer des objets à l'aide de valeurs littérales au lieu d'utiliser l'opérateur `new`. Par exemple, vous pouvez créer un tableau à l'aide d'une valeur littérale ou du constructeur de classe Array, comme suit :

```
var someArray:Array = [1, 2, 3]; // literal value
var someArray:Array = new Array(1,2,3); // Array constructor
```

Vérification des types

La vérification des types peut être effectuée lors de la compilation ou de l'exécution. Les langages typés statiquement (C++ et Java, par exemple) effectuent la vérification des types lors de la compilation. Les langages typés dynamiquement (Smalltalk et Python, par exemple) effectuent la vérification des types lors de l'exécution. En tant que langage typé dynamiquement, ActionScript 3.0 effectue la vérification des types lors de l'exécution mais également lors de la compilation, avec un mode de compilateur spécial appelé *mode strict*. En mode strict, la vérification des types a lieu à la fois lors de la compilation et de l'exécution, mais en mode standard, elle a lieu lors de l'exécution.

Les langages typés dynamiquement vous permettent de structurer votre code avec une grande souplesse, mais des erreurs de type risquent de se produire lors de l'exécution. Les langages typés statiquement signalent des erreurs de type lors de la compilation mais des informations de type sont requises à ce moment-là.

Vérification des types lors de la compilation

La vérification des types lors de la compilation est souvent favorisée dans les projets plus importants car au fur et à mesure que la taille du projet augmente, la flexibilité du type de données devient généralement moins importante que la capture d'erreurs de type aussitôt que possible. C'est pourquoi le compilateur d'ActionScript dans Adobe Flash CS4 Professional et dans Adobe Flex Builder est exécuté en mode strict, par défaut.

Pour que la vérification des types lors de la compilation soit exécutée, le compilateur doit connaître les informations de type de données pour les variables ou les expressions dans votre code. Pour déclarer explicitement un type de données pour une variable, ajoutez l'opérateur deux points (`:`) suivi du type de données comme suffixe du nom de variable. Pour associer un type de données à un paramètre, utilisez l'opérateur deux points suivi du type de données. Par exemple, le code suivant ajoute des informations de type de données au paramètre `xParam`, et déclare une variable `myParam` avec un type de données explicite :

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

En mode strict, le compilateur d'ActionScript signale des incompatibilités de type comme erreurs de compilateur. Par exemple, le code suivant déclare un paramètre de fonction `xParam`, de type Object, mais tente ensuite d'affecter des valeurs de type String et Number à ce paramètre. Ceci produit une erreur de compilateur en mode strict.

```
function dynamicTest(xParam:Object)
{
    if (xParam is String)
    {
        var myStr:String = xParam; // compiler error in strict mode
        trace("String: " + myStr);
    }
    else if (xParam is Number)
    {
        var myNum:Number = xParam; // compiler error in strict mode
        trace("Number: " + myNum);
    }
}
```

Même en mode strict, cependant, vous pouvez choisir d'abandonner la vérification des types lors de la compilation en laissant la partie droite d'une instruction d'affectation non typée. Vous pouvez marquer une variable ou une expression comme non typée soit en omettant une annotation de type, soit à l'aide de l'annotation de type astérisque (`*`) spéciale. Par exemple, si le paramètre `xParam` dans l'exemple précédent est modifié de façon à ne plus avoir d'annotation de type, le code compile en mode strict :

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

Vérification des types lors de l'exécution

La vérification des types lors de l'exécution s'applique dans ActionScript 3.0 quel que soit le mode, strict ou standard. Imaginez que la valeur 3 est transmise en tant qu'argument à une fonction qui attend un tableau. En mode strict, le compilateur génère une erreur car la valeur 3 n'est pas compatible avec le type de données Array. Si vous désactivez le mode strict et utilisez le mode standard, le compilateur ne signale pas l'incompatibilité de type, mais la vérification des types lors de l'exécution effectuée par Flash Player et Adobe AIR provoque une erreur d'exécution.

L'exemple suivant présente une fonction appelée `typeTest()` qui attend une instruction Array mais qui reçoit une valeur de 3. Ceci provoque une erreur d'exécution en mode standard car la valeur 3 n'est pas un membre du type de données déclaré (Array) du paramètre.

```
function typeTest(xParam:Array)
{
    trace(xParam);
}
var myNum:Number = 3;
typeTest(myNum);
// run-time error in ActionScript 3.0 standard mode
```

Il se peut également que vous obteniez une erreur de type lors de l'exécution alors que vous êtes en mode strict. Ceci est possible si vous utilisez le mode strict mais que vous abandonnez la vérification des types lors de la compilation à l'aide d'une variable non typée. Lorsque vous utilisez une variable non typée, vous n'éliminez pas la vérification des types mais la remettez à plus tard, au moment de l'exécution. Par exemple, si la variable `myNum` de l'exemple précédent n'a pas de type de données déclaré, le compilateur ne peut pas détecter l'incompatibilité de type, mais Flash Player et Adobe AIR génèrent une erreur d'exécution car ils comparent la valeur d'exécution de `myNum` (définie sur 3 en raison de l'instruction d'affectation), avec le type de `xParam` (défini sur le type de données Array).

```
function typeTest(xParam:Array)
{
    trace(xParam);
}
var myNum = 3;
typeTest(myNum);
// run-time error in ActionScript 3.0
```

La vérification des types lors de l'exécution permet également d'utiliser l'héritage avec davantage de souplesse que la vérification lors de la compilation. En remettant la vérification des types au moment de l'exécution, le mode standard vous permet de référencer des propriétés d'une sous-classe même si vous effectuez un *upcast*. Un upcast a lieu lorsque vous utilisez une classe de base pour déclarer le type d'une occurrence de classe mais une sous-classe pour l'instancier. Par exemple, vous pouvez créer une classe appelée ClassBase qui peut être étendue (les classes avec l'attribut `final` ne peuvent pas être étendues):

```
class ClassBase
{
}
```

Vous pouvez ensuite créer une sous-classe de ClassBase appelée ClassExtender, qui possède une propriété appelée `someString`, comme suit :

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

Vous pouvez utiliser les deux classes pour créer une occurrence de classe déclarée à l'aide du type de données ClassBase mais instanciée avec le constructeur ClassExtender. Un upcast est considéré comme une opération sûre car la classe de base ne contient aucune propriété ni méthode ne se trouvant pas dans la sous-classe.

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

Une sous-classe, néanmoins, contient des propriétés et des méthodes que sa classe de base ne contient pas. Par exemple, la classe ClassExtender contient la propriété `someString` qui n'existe pas dans la classe ClassBase. Dans ActionScript 3.0 en mode standard, vous pouvez référencer cette propriété à l'aide de l'occurrence `myClass` sans générer d'erreur d'exécution, comme indiqué dans l'exemple suivant :

```
var myClass:ClassBase = new ClassExtender();
myClass.someString = "hello";
// no error in ActionScript 3.0 standard mode
```

Opérateur is

L'opérateur `is` (nouveau dans ActionScript 3.0), vous permet de tester si une variable ou une expression est membre d'un type de données. Dans les versions précédentes d'ActionScript, l'opérateur `instanceof` offrait cette fonctionnalité, mais dans ActionScript 3.0, l'opérateur `instanceof` ne doit pas être utilisé pour tester l'appartenance à un type de données. L'opérateur `is` doit être utilisé à la place de l'opérateur `instanceof` pour la vérification des types manuelle car l'expression `x instanceof y` vérifie simplement la chaîne de prototype de `x` pour voir si `y` existe (et dans ActionScript 3.0, la chaîne de prototype ne donne pas une image complète de la hiérarchie d'héritage).

L'opérateur `is` examine sa hiérarchie d'héritage et peut être utilisé pour vérifier non seulement si un objet est une occurrence d'une classe particulière mais également si un objet est une occurrence d'une classe qui implémente une interface particulière. L'exemple suivant crée une occurrence de la classe Sprite appelée `mySprite` et utilise l'opérateur `is` pour tester si `mySprite` est une occurrence des classes Sprite et DisplayObject, et si elle implémente l'interface IEventDispatcher:

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

L'opérateur `is` vérifie la hiérarchie d'héritage et signale correctement que `mySprite` est compatible avec les classes Sprite et DisplayObject (la classe Sprite est une sous-classe de la classe DisplayObject). L'opérateur `is` vérifie également si `mySprite` hérite d'une classe qui implémente l'interface IEventDispatcher. Etant donné que la classe Sprite hérite de la classe EventDispatcher, qui implémente l'interface IEventDispatcher, l'opérateur `is` signale correctement que `mySprite` implémente la même interface.

L'exemple suivant présente les mêmes tests que l'exemple précédent, mais avec `instanceof` au lieu de l'opérateur `is`. L'opérateur `instanceof` identifie correctement que `mySprite` est une occurrence de Sprite ou de DisplayObject, mais il renvoie `false` lorsqu'il est utilisé pour tester si `mySprite` implémente l'interface IEventDispatcher.

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

Opérateur as

L'opérateur `as` (nouveau dans ActionScript 3.0), vous permet également de vérifier si une expression est membre d'un type de données. Contrairement à l'opérateur `is`, cependant, l'opérateur `as` ne renvoie pas de valeur booléenne. L'opérateur `as` renvoie la valeur de l'expression au lieu de `true`, et `null` au lieu de `false`. L'exemple suivant indique les résultats obtenus si vous utilisez l'opérateur `as` au lieu de l'opérateur `is` pour vérifier simplement si une occurrence de Sprite appartient aux types de données DisplayObject, IEventDispatcher et Number.

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

Lorsque vous utilisez l'opérateur `as`, l'opérande à droite doit être un type de données. Vous obtenez une erreur si vous tentez d'utiliser une expression autre qu'un type de données comme opérande à droite.

Classes dynamiques

Une classe *dynamique* définit un objet qui peut être modifié lors de l'exécution en ajoutant ou en modifiant des propriétés et des méthodes. Une classe qui n'est pas dynamique (la classe String, par exemple), est une classe *scellée*. Vous ne pouvez pas ajouter de propriétés ni de méthodes à une classe scellée lors de l'exécution.

Vous créez des classes dynamiques en utilisant l'attribut `dynamic` lorsque vous déclarez une classe. Par exemple, le code suivant crée une classe dynamique appelée `Protean` :

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

Si vous instanciez ensuite une occurrence de la classe `Protean`, vous pouvez lui ajouter des propriétés ou des méthodes en dehors de la définition de classe. Par exemple, le code suivant crée une occurrence de la classe `Protean` et lui ajoute une propriété appelée `aString` et une autre appelée `aNumber` :

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

Les propriétés que vous ajoutez à une occurrence d'une classe dynamique sont des entités d'exécution. Par conséquent, toute vérification des types est effectuée lors de l'exécution. Vous ne pouvez pas ajouter une annotation de type à une propriété que vous ajoutez de cette façon.

Vous pouvez également ajouter une méthode à l'occurrence `myProtean` en définissant une fonction et en l'associant à une propriété de l'occurrence `myProtean`. Le code suivant déplace l'instruction trace dans une méthode appelée `traceProtean()` :

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

Cependant, les méthodes créées de cette manière n'ont pas accès à des méthodes ou à des propriétés privées de la classe Protean. De plus, même les références à des méthodes ou à des propriétés publiques de la classe `Protean` doivent être qualifiées avec le mot-clé `this` ou le nom de classe. L'exemple suivant présente la méthode `traceProtean()` tentant d'accéder aux variables privées et publiques de la classe `Protean`.

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

Descriptions des types de données

Les types de données primitifs peuvent être Boolean, int, Null, Number, String, uint et void. Les classes de base d'ActionScript définissent également les types de données complexes suivants : Object, Array, Date, Error, Function, RegExp, XML et XMLList.

Type de données Boolean

Le type de données Boolean comporte deux valeurs :`true` et `false`. Les variables du type booléen ne prennent en charge aucune autre valeur. La valeur par défaut d'une variable booléenne qui a été déclarée mais non initialisée est `false`.

Type de données int

Le type de données int est stocké en interne en tant que nombre entier 32 bits et comprend l'ensemble des entiers allant de

-2 147 483 648 (-2^{31}) à 2 147 483 647 (2^{31} - 1), inclus. Les versions précédentes d'ActionScript offraient uniquement le type de données Number, qui était utilisé à la fois pour les nombres entiers et les nombres en virgule flottante. Dans ActionScript 3.0, vous avez maintenant accès à des types machine de bas niveau pour les nombres entiers 32 bits signés et non signés. Si votre variable ne doit pas utiliser de nombres en virgule flottante, l'utilisation du type de données int au lieu du type de données Number est plus rapide et plus efficace.

Pour les valeurs entières en dehors de la plage des valeurs int minimum et maximum, utilisez le type de données Number, qui peut gérer des valeurs positives et négatives 9 007 199 254 740 992 (valeurs entières 53 bits). La valeur par défaut pour des variables du type de données int est 0.

Type de données Null

Le type de données Null contient une seule valeur, `null`. Il s'agit de la valeur par défaut pour le type de données String et toutes les classes qui définissent des types de données complexes, y compris la classe Object. Aucun des autres types de données primitifs (Boolean, Number, int et uint) ne contient la valeur `null`. Flash Player et Adobe AIR convertissent la valeur `null` en la valeur par défaut appropriée si vous tentez d'affecter `null` à des variables de type Boolean, Number, int, ou uint. Vous ne pouvez pas utiliser ce type de donnés comme annotation de type.

Type de données Number

Dans ActionScript 3.0, le type de données Number peut représenter des entiers, des entiers non signés et des nombres en virgule flottante. Néanmoins, pour optimiser les performances, utilisez le type de données Number uniquement pour des valeurs entières supérieures à ce que les types `int` et `uint` 32 bits peuvent stocker ou pour des nombres en virgule flottante. Pour stocker un nombre en virgule flottante, vous devez y insérer une virgule. Si vous omettez la virgule, le nombre est stocké comme nombre entier.

Le type de données Number utilise le format à deux décimales (64 bits) conformément à la norme IEEE 754. Cette norme dicte comment les nombres en virgule flottante sont stockés à l'aide des 64 bits disponibles. Un bit est utilisé pour désigner si le nombre est positif ou négatif. Onze bits sont utilisés pour l'exposant, qui est stocké comme base 2. Les 52 bits restants sont utilisés pour stocker le *significande* (également appelé *mantisse*), qui correspond au nombre élevé à la puissance indiqué par l'exposant.

En utilisant certains de ses bits pour stocker un exposant, le type de données Number peut stocker des nombres en virgule flottante beaucoup plus grands que s'il utilisait tous ses bits pour le significande. Par exemple, si le type de données Number utilisait les 64 bits pour stocker le significande, il pourrait stocker un nombre aussi grand que 2^{65} - 1. En utilisant 11 bits pour stocker un exposant, le type de données Number peut élever son significande à une puissance de 2^{1023}.

Les valeurs maximum et minimum que le type Number peut représenter sont stockées dans des propriétés statiques de la classe Number appelées `Number.MAX_VALUE` et `Number.MIN_VALUE`.

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

Bien que cette plage de nombres soit énorme, elle est d'une grande précision. Le type de données Number utilise 52 bits pour stocker le significande. Par conséquent, les nombres nécessitant plus de 52 bits pour une représentation précise (la fraction 1/3, par exemple) ne sont que des approximations. Si votre application exige une précision absolue avec des nombres décimaux, vous devez utiliser le logiciel qui implémente l'arithmétique flottante décimale plutôt que l'arithmétique flottante binaire.

Lorsque vous stockez des valeurs entières avec le type de données Number, seuls les 52 bits du significande sont utilisés. Le type de données Number utilise ces 52 bits et un bit masqué spécial pour représenter des nombres entiers de -9 007 199 254 740 992 (-2^{53}) à 9 007 199 254 740 992 (2^{53}).

Flash Player et Adobe AIR utilisent la valeur `NaN` non seulement comme valeur par défaut pour des variables de type `Number`, mais également comme résultat de toute opération devant renvoyer un nombre mais ne le faisant pas. Par exemple, si vous tentez de calculer la racine carrée d'un nombre négatif, le résultat est `NaN`. D'autres valeurs Number spéciales comprennent l'*infini positif* et l'*infini négatif*.

Remarque : *le résultat de la division par `0` n'est `NaN` que si le diviseur est également `0`. La division par `0` produit `infinity` lorsque le dividende est positif ou `-infinity` lorsqu'il est négatif.*

Type de données String

Le type de données String représente une séquence de caractères 16 bits. Les chaînes sont stockées en interne sous forme de caractères Unicode, au format UTF-16. Les chaînes sont des valeurs inaltérables, comme dans le langage de programmation Java. Toute opération effectuée sur une valeur String renvoie une nouvelle occurrence de la chaîne. La valeur par défaut d'une variable déclarée avec le type de données String est `null`. La valeur `null` n'est pas identique à la chaîne vide (`""`), même si elles représentent toutes les deux l'absence de caractères.

Type de données uint

Le type de données uint est stocké en interne sous la forme d'un entier 32 bits non signé et est constitué de l'ensemble d'entiers compris entre 0 et 4 294 967 295 (2^{32}-1), inclus. Utilisez le type de données uint dans des situations spéciales nécessitant des entiers non négatifs. Par exemple, vous devez utiliser le type de données uint pour représenter les valeurs de couleur des pixels car le type de données int a un bit de signe interne non approprié pour la gestion des valeurs de couleur. Pour les valeurs d'entiers supérieures à la valeur uint maximale, utilisez le type de données Number qui peut gérer des valeurs d'entiers 53 bits. La valeur par défaut pour des variables du type de données uint est 0.

Type de données Void

Le type de données void contient une seule valeur, `undefined`. Dans les versions précédentes d'ActionScript, `undefined` était la valeur par défaut des occurrences de la classe Object. Dans ActionScript 3.0, la valeur par défaut des occurrences Object est `null`. Si vous tentez d'affecter la valeur `undefined` à une occurrence de la classe Object, Flash Player ou Adobe AIR convertit la valeur en `null`. Vous pouvez affecter uniquement une valeur de `undefined` à des variables non typées. Les variables non typées sont des variables dépourvues de toute annotation de type, ou qui utilisent l'astérisque (`*`) pour l'annotation de type. Vous pouvez utiliser `void` uniquement comme annotation de type renvoyé.

Type de données Object

Ce type de données est défini par la classe Object. La classe Object sert de classe de base pour toutes les définitions de classe dans ActionScript. La version d'ActionScript 3.0 du type de données Object diffère de celle des versions précédentes de trois façons. Premièrement, le type de données Object n'est plus le type de données par défaut affecté aux variables sans annotation de type. Deuxièmement, le type de données Object ne comprend plus la valeur `undefined`, qui était la valeur par défaut des occurrences Object. Troisièmement, dans ActionScript 3.0, la valeur par défaut des occurrences de la classe Object est `null`.

Dans les versions précédentes d'ActionScript, une variable sans annotation de type était affectée automatiquement au type de données Object. Ceci n'est plus valable dans ActionScript 3.0, qui inclut maintenant la notion de variable parfaitement non typée. Les variables sans annotation de type sont désormais considérées comme non typées. Si vous souhaitez faire comprendre aux lecteurs de votre code que vous souhaitez laisser une variable non typée, vous pouvez utiliser le nouvel astérisque (`*`) pour l'annotation de type, ce qui revient à omettre une annotation de type. L'exemple suivant présente deux instructions équivalentes qui déclarent une variable non typée `x` :

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

Seules les variables non typées peuvent contenir la valeur `undefined`. Si vous tentez d'affecter la valeur `undefined` à une variable ayant un type de données, Flash Player ou Adobe AIR convertit la valeur `undefined` en la valeur par défaut pour ce type de données. Pour des occurrences du type de données Object, la valeur par défaut est `null`, ce qui signifie que Flash Player ou Adobe AIR convertit la valeur `undefined` en `null` si vous tentez d'affecter `undefined` à une occurrence Object.

Conversions de type

Une conversion de type a lieu lorsqu'une valeur est transformée en une valeur d'un type de données différent. Les conversions de type peuvent être *implicites* ou *explicites*. Les conversions implicites (ou *coercition*) sont parfois effectuées par Flash Player ou Adobe AIR lors de l'exécution. Par exemple, si la valeur 2 est affectée à une variable du type de données Boolean, Flash Player ou Adobe AIR convertit la valeur 2 en la valeur booléenne `true` avant de l'affecter à la variable. Les conversions explicites (ou *association*) ont lieu lorsque votre code demande au compilateur de traiter une variable d'un type de données comme si elle appartenait à un type de données différent. Lorsque des valeurs primitives sont impliquées, l'association convertit les valeurs d'un type de données en un autre. Pour associer un objet à un autre type de données, vous mettez le nom de l'objet entre parenthèses et le faites précéder du nom du nouveau type. Par exemple, le code suivant prend une valeur booléenne et la transforme en entier numérique:

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

Conversions implicites

Les conversions implicites ont lieu lors de l'exécution dans plusieurs contextes :

- Dans des instructions d'affectation
- Lorsque des valeurs sont transmises en tant qu'arguments de fonction
- Lorsque des valeurs sont renvoyées à partir de fonctions
- Dans les expressions utilisant certains opérateurs tels que l'opérateur d'addition (`+`)

  Pour les types définis par l'utilisateur, les conversions implicites ont lieu lorsque la valeur à convertir est une occurrence de la classe de destination ou d'une classe qui dérive de cette dernière. En cas d'échec de la conversion implicite, une erreur se produit. Par exemple, le code suivant contient une conversion implicite ayant réussi et une conversion implicite ayant échoué:

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

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

objA = objB; // Conversion succeeds.
objB = arr; // Conversion fails.
```

Pour les types primitifs, les conversions implicites sont gérées en appelant les mêmes algorithmes de conversion internes appelés par les fonctions de conversion explicite. La section suivante traite de ces conversions de type primitif dans les détails.

Conversions explicites

Les conversions explicites, ou association, sont utiles lorsque vous compilez en mode strict et que vous ne souhaitez pas qu'une incompatibilité de types génère une erreur de compilation. Ceci peut se produire lorsque vous savez que la coercition convertira vos valeurs correctement lors de l'exécution. Par exemple, lorsque vous utilisez des données reçues d'un formulaire, vous pouvez utiliser la coercition pour convertir certaines valeurs de chaîne en valeurs numériques. Le code suivant génère une erreur de compilation même si le code s'exécute correctement en mode standard :

```
var quantityField:String = "3";
var quantity:int = quantityField; // compile time error in strict mode
```

Si vous souhaitez continuer à utiliser le mode strict, mais que vous souhaitez que la chaîne soit convertie en nombre entier, vous pouvez utiliser la conversion explicite, comme suit :

```
var quantityField:String = "3";
var quantity:int = int(quantityField); // Explicit conversion succeeds.
```

Association à int, uint et Number

Vous pouvez associer tout type de données à l'un des trois types de nombre suivants : int, uint et Number. Si Flash Player ou Adobe AIR ne peut pas convertir le nombre, la valeur par défaut 0 est affectée pour les types de données int et uint, et la valeur par défaut `NaN` est affectée pour le type de données Number. Si vous convertissez une valeur booléenne en un nombre, `true` devient la valeur 1 et `false` devient la valeur 0.

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

Les valeurs de chaîne qui contiennent des chiffres uniquement peuvent être converties en l'un des types de nombre. Les types de nombre peuvent également convertir des chaînes ressemblant à des nombres négatifs ou des chaînes représentant une valeur hexadécimale (par exemple, `0x1A`). Le processus de conversion ignore les espaces blancs à gauche ou à droite dans la valeur de chaîne. Vous pouvez également associer des chaînes qui ressemblent à des nombres en virgule flottante à l'aide de `Number()`. Le fait d'inclure une virgule fait que `uint()` et `int()` renvoient un entier avec les caractères suivant la virgule ayant été tronqués. Par exemple, les valeurs de chaîne suivantes peuvent être associées à des nombres :

```
trace(uint("5")); // 5
trace(uint("-5")); // 4294967291. It wraps around from MAX_VALUE
trace(uint(" 27 ")); // 27
trace(uint("3.7")); // 3
trace(int("3.7")); // 3
trace(int("0x1A")); // 26
trace(Number("3.7")); // 3.7
```

Les valeurs de chaîne qui contiennent des caractères non numériques renvoient 0 lors de l'association à `int()` ou `uint()` et `NaN` lors de l'association à `Number()`. Le processus de conversion ignore les espaces blancs à gauche et à droite mais renvoie 0 ou `NaN` si une chaîne contient un espace blanc séparant deux nombres.

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

Dans ActionScript 3.0, la fonction `Number()` ne prend plus en charge les nombres octaux, ou de base 8. Si vous fournissez une chaîne avec un zéro à gauche à la fonction `Number()` d'ActionScript 2.0, le nombre est interprété comme un nombre octal et converti en son équivalent décimal. Ceci ne s'applique pas à la fonction `Number()` dans ActionScript 3.0, qui ignore le zéro à gauche. Par exemple, le code suivant génère un résultat différent lorsqu'il est compilé à l'aide de différentes versions d'ActionScript :

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

L'association n'est pas nécessaire lorsqu'une valeur d'un type numérique est affectée à une variable d'un type numérique différent. Même en mode strict, les types numériques sont convertis de façon implicite en d'autres types numériques. Ceci signifie que dans certains cas, des valeurs inattendues peuvent être renvoyées lorsque la plage d'un type est dépassée. Les exemples suivants compilent tous en mode strict, même si certains génèrent des valeurs inattendues :

```
var myUInt:uint = -3; // Assign int/Number value to uint variable
trace(myUInt); // 4294967293

var myNum:Number = sampleUINT; // Assign int/uint value to Number variable
trace(myNum) // 4294967293

var myInt:int = uint.MAX_VALUE + 1; // Assign Number value to uint variable
trace(myInt); // 0

myInt = int.MAX_VALUE + 1; // Assign uint/Number value to int variable
trace(myInt); // -2147483648
```

Le tableau suivant récapitule les résultats de l'association au type de données Number, int, ou uint à partir d'autres types de données.

Valeur ou type de données	**Résultat de la conversion en Number, int ou uint**
Boolean	Si la valeur est `true`, 1 ; sinon, 0.
Date	Représentation interne de l'objet Date, qui est le nombre de millisecondes depuis le 1er janvier 1970, à minuit, heure universelle.
`null`	0

Valeur ou type de données	Résultat de la conversion en Number, int ou uint
Object	Si l'occurrence est `null` et convertie en Number, `NaN` ; sinon, 0.
String	Un nombre si Flash Player ou Adobe AIR peut convertir la chaîne en un nombre ; autrement, `NaN` si converti en Number ou 0 si converti en int ou uint.
`undefined`	Si converti en Number, `NaN` ; si converti en int ou uint, 0.

Association à Boolean

L'association à Boolean à partir de n'importe quel type de données numériques (uint, int et Number) donne `false` si la valeur numérique est 0, et `true` autrement. Pour le type de données Number, la valeur `NaN` donne également `false`. L'exemple suivant indique les résultats de l'association des chiffres -1, 0, et 1 :

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

Le résultat de l'exemple indique que sur les trois chiffres, seul 0 renvoie une valeur `false`:

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

L'association à Boolean à partir d'une valeur String renvoie `false` si la chaîne est `null` ou une chaîne vide (`""`). Sinon, elle renvoie `true`.

```
var str1:String; // Uninitialized string is null.
trace(Boolean(str1)); // false

var str2:String = ""; // empty string
trace(Boolean(str2)); // false

var str3:String = " "; // white space only
trace(Boolean(str3)); // true
```

L'association à Boolean à partir d'une occurrence de la classe Object renvoie `false` si l'occurrence est `null`, et `true` autrement :

```
var myObj:Object; // Uninitialized object is null.
trace(Boolean(myObj)); // false

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

Les variables booléennes bénéficient d'un traitement particulier en mode strict car vous pouvez affecter des valeurs de tout type de données à une variable booléenne sans association. La coercition implicite de tous les types de données au type de données Boolean a lieu même en mode strict. En d'autres termes, contrairement à la plupart de tous les autres types de données, l'association à Boolean n'est pas nécessaire pour éviter des erreurs en mode strict. Les exemples suivants compilent tous en mode strict et se comportent comme prévu lors de l'exécution :

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

Le tableau suivant récapitule les résultats de l'association au type de données Boolean à partir d'autres types de données :

Valeur ou type de données	Résultat de la conversion en Boolean
String	`false` si la valeur est `null` ou la chaîne vide (`""`) ; `true` autrement.
`null`	`false`
Number, int ou uint	`false` si la valeur est `NaN` ou 0 ; `true` autrement.
Object	`false` si l'occurrence est `null` ; `true` autrement.

Association à String

L'association au type de données String à partir de n'importe quel type de données numérique renvoie une représentation sous forme de chaîne du nombre. L'association au type de données String à partir d'une valeur booléenne renvoie la chaîne `"true"` si la valeur est `true`, et renvoie la chaîne `"false"` si la valeur est `false`.

L'association au type de données String à partir d'une occurrence de la classe Object renvoie la chaîne `"null"` si l'occurrence est `null`. Autrement, l'association au type String à partir de la classe Object renvoie la chaîne `"[object Object]"` .

L'association à String à partir d'une occurrence de la classe Array renvoie une chaîne comprenant une liste séparée par des virgules de tous les éléments du tableau. Par exemple, l'association suivante au type de données String renvoie une chaîne contenant les trois éléments du tableau :

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

L'association à String à partir d'une occurrence de la classe Date renvoie une représentation sous forme de chaîne de la date que l'occurrence contient. Par exemple, l'exemple suivant renvoie une représentation sous forme de chaîne de l'occurrence de la classe Date (le résultat indiqué correspond à l'heure d'été de la côte ouest des Etats-Unis) :

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

Le tableau suivant récapitule les résultats de l'association au type de données String à partir d'autres types de données.

Valeur ou type de données	Résultat de la conversion en chaîne
Array	Une chaîne comprenant tous les éléments du tableau
Boolean	`"true"` ou `"false"`
Date	Une représentation sous forme de chaîne de l'objet Date

Valeur ou type de données	Résultat de la conversion en chaîne
`null`	`"null"`
Number, int ou uint	Une représentation sous forme de chaîne du nombre
Object	Si l'occurrence est null, `"null"` ; sinon, `"[object Object]"`.

Syntaxe

La syntaxe d'un langage définit un ensemble de règles à suivre lors de l'écriture du code exécutable.

Respect de la casse

ActionScript 3.0 est un langage qui fait la distinction entre les majuscules et les minuscules. Les identifiants qui diffèrent au niveau de la casse uniquement sont considérés comme différents. Par exemple, le code suivant crée deux variables différentes :

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

Syntaxe à point

L'opérateur point (.) permet d'accéder aux propriétés et aux méthodes d'un objet. La syntaxe à point vous permet de vous rapporter à une méthode ou à une propriété de classe à l'aide d'un nom d'occurrence, suivi par l'opérateur point et le nom de la méthode ou de la propriété. Par exemple, considérez la définition de classe suivante :

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

La syntaxe à point vous permet d'accéder à la propriété `prop1` et à la méthode `method1()` à l'aide du nom d'occurrence créé dans le code suivant :

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

Vous pouvez utiliser la syntaxe à point lorsque vous définissez des packages. Vous utilisez l'opérateur point pour référencer des packages imbriqués. Par exemple, la classe EventDispatcher se trouve dans un package appelé events qui est imbriqué dans le package nommé flash. Vous pouvez référencer le package events à l'aide de l'expression suivante :

```
flash.events
```

Vous pouvez également référencer la classe EventDispatcher au moyen de cette expression :

```
flash.events.EventDispatcher
```

Syntaxe à barre oblique

La syntaxe à barre oblique n'est pas prise en charge dans ActionScript 3.0. Elle était utilisée dans les versions antérieures d'ActionScript pour indiquer le chemin d'un clip ou d'une variable.

Littéraux

Un *littéral* est une valeur qui apparaît directement dans votre code. Voici quelques exemples de littéraux :

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

Les littéraux peuvent également être regroupés pour former des littéraux composés. Les littéraux de tableau sont placés entre crochets (`[]`) et utilisent la virgule pour séparer les éléments du tableau.

Un littéral de tableau permet d'initialiser un tableau. Les exemples suivants présentent deux tableaux initialisés par des littéraux de tableau. Vous pouvez utiliser l'instruction `new` et transmettre le littéral composé sous forme de paramètre au constructeur de classe Array, ou affecter directement les valeurs littérales lors de l'instanciation des occurrences des classes de base ActionScript suivantes : Object, Array, String, Number, int, uint, XML, XMLList et Boolean.

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

// Assign literal directly.
var myStrings:Array = ["alpha", "beta", "gamma"];
var myNums:Array = [1,2,3,5,8];
```

Les littéraux permettent également d'initialiser un objet générique. Un objet générique est une occurrence de la classe Object. Les littéraux d'objet sont placés entre accolades (`{}`) et utilisent la virgule pour séparer les propriétés de l'objet. Chaque propriété est déclarée avec le caractère deux-points (`:`), séparant le nom et la valeur de la propriété.

Vous pouvez créer un objet générique via l'instruction `new` et transmettre le littéral d'objet sous forme de paramètre au constructeur de classe Object ou affecter directement le littéral d'objet à l'occurrence déclarée. L'exemple suivant illustre deux méthodes différentes de créer un objet générique et de l'initialiser avec trois propriétés (`propA`, `propB` et `propC`), chacune avec des valeurs définies sur 1, 2, et 3, respectivement :

```
// Use new statement and add properties.
var myObject:Object = new Object();
myObject.propA = 1;
myObject.propB = 2;
myObject.propC = 3;

// Assign literal directly.
var myObject:Object = {propA:1, propB:2, propC:3};
```

Pour plus d'informations, consultez les sections « Principes de base des chaînes » à la page 144, « Principes de base des expressions régulières » à la page 211 et « Initialisation de variables XML » à la page 240.

Points-virgules

Vous pouvez utiliser le point-virgule (`;`) à la fin d'une instruction. Si vous omettez ce caractère, le compilateur suppose que chaque ligne de code représente une instruction distincte. Etant donné que de nombreux programmeurs ont l'habitude d'utiliser le point-virgule pour marquer la fin d'une instruction, vous pouvez faciliter la lecture de votre code en procédant de la sorte.

L'ajout d'un point-virgule à la fin d'une instruction permet de placer plusieurs instructions sur une même ligne, mais rend la lecture de votre code plus difficile.

Parenthèses

Vous pouvez utiliser les parenthèses (`()`) de trois façons dans ActionScript 3.0. Premièrement, elles permettent de modifier l'ordre des opérations dans une expression. Les opérations regroupées à l'intérieur de parenthèses sont toujours exécutées en premier lieu. Par exemple, vous pouvez utiliser des parenthèses pour modifier l'ordre des opérations dans le code suivant :

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

Deuxièmement, vous pouvez utiliser des parenthèses avec l'opérateur virgule (`,`) pour évaluer une série d'expressions et renvoyer le résultat de l'expression finale, comme indiqué dans l'exemple suivant :

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

Troisièmement, vous pouvez utiliser des parenthèses pour transmettre un ou plusieurs paramètres à des fonctions ou à des méthodes, comme indiqué dans l'exemple suivant, qui transmet une valeur String à la fonction `trace()` :

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

Commentaires

Le code ActionScript 3.0 prend en charge deux types de commentaires : les commentaires d'une ligne et les commentaires de plusieurs lignes. Ces mécanismes de commentaire sont semblables aux mécanismes de commentaire de C++ et Java. Le compilateur ignore le texte marqué comme commentaire.

Les commentaires d'une ligne commencent par deux barres obliques (`//`) et continuent jusqu'à la fin de la ligne. Par exemple, le code suivant contient un commentaire d'une ligne :

```
var someNumber:Number = 3; // a single line comment
```

Les commentaires de plusieurs lignes commencent par une barre oblique et un astérisque (`/*`) et se terminent pas un astérisque et une barre oblique (`*/`).

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

Mots-clés et mots réservés

Les *mots réservés* ne peuvent pas servir d'identifiants dans votre code car leur utilisation est réservée à ActionScript. Les mots réservés comprennent les *mots lexicaux* qui sont supprimés de l'espace de noms du programme par le compilateur. Le compilateur signale une erreur si vous utilisez un mot-clé lexical comme identifiant. Le tableau suivant répertorie les mots lexicaux d'ActionScript 3.0.

as	break	case	catch
classe	const	continue	default
delete	do	else	extends
false	finally	for	fonction
if	implements	import	in

instanceof	interface	internal	is
native	new	null	package
private	protected	public	return
super	switch	this	throw
to	true	try	typeof
use	var	void	while
with			

Il existe un petit ensemble de mots-clés, appelés *mots-clés syntaxiques* qui peuvent servir d'identifiants, mais qui possèdent une signification spéciale dans certains contextes. Le tableau suivant répertorie les mots-clés syntaxiques d'ActionScript 3.0.

each	get	set	namespace
inclure	dynamique	final	native
override	static		

Il existe également plusieurs identifiants auxquels il est parfois fait référence sous le terme *mots réservés pour une utilisation future*. Ces identifiants ne sont pas réservés par ActionScript 3.0, même si certains risquent d'être considérés comme des mots-clés par le logiciel incorporant ActionScript 3.0. Vous pourrez peut-être utiliser un grand nombre d'entre eux dans votre code, mais Adobe vous le déconseille car il est possible qu'ils soient utilisés en tant que mots-clés dans une version ultérieure du langage.

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

Constantes

ActionScript 3.0 prend en charge l'instruction `const` que vous pouvez utiliser pour créer des constantes. Les constantes sont des propriétés dont la valeur est fixe et non modifiable. Vous pouvez affecter une valeur à une constante une seule fois, et l'affectation doit avoir lieu à proximité de la déclaration de la constante. Par exemple, si une constante est déclarée en tant que membre d'une classe, vous pouvez lui affecter une valeur uniquement dans la déclaration ou dans le constructeur de classe.

Le code suivant déclare deux constantes. La première constante, `MINIMUM`, a une valeur affectée dans l'instruction de déclaration. La seconde constante, `MAXIMUM`, a une valeur affectée dans le constructeur. Observez que cet exemple compile uniquement en mode standard car, en mode strict, il est uniquement possible d'affecter une valeur de constante lors de l'initialisation.

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

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

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

Une erreur se produit si vous tentez d'affecter une valeur initiale à une constante de toute autre façon. Par exemple, si vous tentez de définir la valeur initiale de `MAXIMUM` en dehors de la classe, une erreur d'exécution se produit.

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

var a:A = new A();
a["MAXIMUM"] = 10; // run-time error
```

ActionScript 3.0 définit un large éventail de constantes à votre usage. Par convention, les constantes dans ActionScript utilisent toutes des majuscules, avec des mots séparés par le caractère de soulignement (_). Par exemple, la définition de classe MouseEvent utilise cette convention d'appellation pour ses constantes, dont chacune représente un événement lié à une action de la souris:

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

Opérateurs

Les opérateurs sont des fonctions spéciales qui utilisent un ou plusieurs opérandes et renvoient une valeur. Un *opérande* est une valeur (généralement un littéral, une variable ou une expression) utilisée par l'opérateur comme entrée. Par exemple, dans le code suivant, les opérateurs d'addition (`+`) et de multiplication (`*`) sont utilisés avec trois opérandes littéraux (2, 3 et 4) pour renvoyer une valeur. Cette valeur est ensuite utilisée par l'opérateur d'affectation (`=`) pour attribuer la valeur renvoyée, 14, à la variable `sumNumber`.

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

Les opérateurs peuvent être unaires, binaires ou ternaires. Un opérateur *unaire* utilise un opérande. Par exemple, l'opérateur d'incrémentation (`++`) est un opérateur unaire car il utilise un seul opérande. Un opérateur *binaire* utilise deux opérandes. Par exemple, l'opérateur de division (`/`) utilise deux opérandes. Un opérateur *ternaire* utilise trois opérandes. Par exemple, l'opérateur conditionnel (`?:`) utilise trois opérandes.

Certains opérateurs sont *surchargés*, ce qui signifie qu'ils se comportent différemment selon le type ou la quantité d'opérandes qui leur est transmis. L'opérateur d'addition (`+`) est un exemple d'opérateur surchargé qui se comporte différemment selon le type de données des opérandes. Si les deux opérandes sont des nombres, l'opérateur d'addition renvoie la somme des valeurs. Si les deux opérandes sont des chaînes, l'opérateur d'addition renvoie la concaténation des deux opérandes. L'exemple de code suivant indique comment l'opérateur se comporte différemment selon les opérandes :

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

Les opérateurs peuvent également se comporter différemment selon le nombre d'opérandes fourni. L'opérateur de soustraction (`-`) est un opérateur à la fois unaire et binaire. Lorsqu'il est fourni avec un seul opérande, l'opérateur de soustraction convertit l'opérande en valeur négative et renvoie le résultat. Lorsqu'il est fourni avec deux opérandes, l'opérateur de soustraction renvoie la différence entre les opérandes. L'exemple suivant présente l'opérateur de soustraction utilisé d'abord comme opérateur unaire, puis comme opérateur binaire.

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

Priorité et associativité des opérateurs

La priorité et l'associativité des opérateurs déterminent leur ordre de traitement. Bien qu'il soit évident, pour ceux qui connaissent bien l'arithmétique, que le compilateur traite l'opérateur de multiplication (`*`) avant l'opérateur d'addition (`+`), le compilateur a besoin d'instructions claires quant à l'ordre à appliquer aux opérateurs. L'ensemble de ces instructions est appelé *ordre de priorité des opérateurs*. ActionScript définit une priorité par défaut que l'opérateur parenthèses (`()`) permet de modifier. Par exemple, le code suivant modifie la priorité par défaut de l'exemple précédent pour forcer le compilateur à traiter l'opérateur d'addition avant l'opérateur de multiplication :

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

Il arrive que plusieurs opérateurs de même priorité apparaissent dans la même expression. Dans ce cas, le compilateur utilise les règles d'*associativité* pour identifier le premier opérateur à traiter. Tous les opérateurs binaires, sauf les opérateurs d'affectation, sont *associatifs gauche*, ce qui signifie que les opérateurs de gauche sont traités avant ceux de droite. Les opérateurs d'affectation et l'opérateur conditionnel (`?:`) sont *associatifs droit*, ce qui signifie que les opérateurs de droite sont traités avant ceux de gauche.

Prenons par exemple les opérateurs inférieur à (`<`) et supérieur à (`>`), qui ont le même ordre de priorité. Lorsque les deux opérateurs sont employés dans la même expression, celui de gauche est traité en premier puisque tous deux sont associatifs gauche. Cela signifie que les deux instructions suivantes donnent le même résultat :

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

L'opérateur supérieur à est traité en premier, ce qui donne une valeur `true` car l'opérande 3 est supérieur à l'opérande 2. La valeur `true` est ensuite transmise à l'opérateur inférieur à, avec l'opérande 1. Le code suivant représente cet état intermédiaire :

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

L'opérateur Inférieur à convertit la valeur `true` en la valeur numérique 1 et compare cette valeur numérique au second opérande 1 pour renvoyer la valeur `false` (la valeur 1 n'est pas inférieure à 1).

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

Vous pouvez modifier l'associativité gauche par défaut avec l'opérateur parenthèses. Vous pouvez demander au compilateur de traiter l'opérateur inférieur à en premier lieu en plaçant cet opérateur et son opérande entre parenthèses. L'exemple suivant utilise l'opérateur parenthèses pour produire un résultat différent en utilisant les mêmes nombres que l'exemple précédent :

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

L'opérateur inférieur à est traité en premier, ce qui donne une valeur `false` car l'opérande 2 n'est pas inférieur à l'opérande 1. La valeur `false` est ensuite transmise à l'opérateur supérieur à, avec l'opérande 3. Le code suivant représente cet état intermédiaire :

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

L'opérateur supérieur à convertit la valeur `false` en la valeur numérique 0 et compare cette valeur numérique à l'autre opérande 3 pour renvoyer la valeur `true` (la valeur 3 est supérieure à 0).

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

Le tableau suivant répertorie les opérateurs gérés par ActionScript 3.0 par ordre de priorité décroissant. Chaque ligne du tableau contient des opérateurs ayant la même priorité. Chaque ligne d'opérateurs a une priorité supérieure à la ligne située au-dessous dans le tableau.

Associer	**Opérateurs**
Principal	`[] {x:y} () f(x) new x.y x[y] <></> @ :: ..`
Suffixe	`x++ x--`
Unaire	`++x --x + - ~ ! delete typeof void`
Multiplication	`* / %`
Ajout	`+ -`
Décalage au niveau du bit	`<< >> >>>`
Relationnel	`< > <= >= as in instanceof is`
Egalité	`== != === !==`
AND au niveau du bit	`&`
XOR au niveau du bit	`^`
OR au niveau du bit	`|`
AND logique	`&&`
OR logique	`||`
Conditionnel	`?:`
Affectation	`= *= /= %= += -= <<= >>= >>>= &= ^= |=`
Virgule	`,`

Opérateurs principaux

Les opérateurs principaux comprennent ceux utilisés pour créer des littéraux Array et Object, regrouper des expressions, appeler des fonctions, instancier des occurrences de classe et accéder à des propriétés.

Tous les opérateurs principaux, comme répertoriés dans le tableau suivant, ont la même priorité. La mention (E4X) apparaît en regard des opérateurs qui font partie de la spécification E4X.

Opérateur	Opération effectuée
`[]`	Initialise un tableau
`{x:y}`	Initialise un objet
`()`	Regroupe des expressions
`f(x)`	Appelle une fonction
`new`	Appelle un constructeur
`x.y x[y]`	Accède à une propriété
`<></>`	Initialise un objet XMLList (E4X)
`@`	Accède à un attribut (E4X)
`::`	Qualifie un nom (E4X)
`..`	Accède à un élément XML descendant (E4X)

Opérateurs de suffixe

Les opérateurs de suffixe prennent un opérateur et incrémentent ou décrémentent sa valeur. Bien que ces opérateurs soient des opérateurs unaires, ils sont classés à part du fait de leur priorité supérieure et de leur comportement particulier. Lorsque vous utilisez un opérateur de suffixe dans une expression plus grande, la valeur de l'expression est renvoyée avant le traitement de cet opérateur. Par exemple, le code suivant montre comment la valeur de l'expression `xNum++` est renvoyée avant l'incrémentation de la valeur :

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

Tous les opérateurs de suffixe, comme répertoriés dans le tableau suivant, ont la même priorité :

Opérateur	Opération effectuée
`++`	Incrémentation (suffixe)
`--`	Décrémentation (suffixe)

Opérateurs unaires

Les opérateurs unaires prennent un opérande. Les opérateurs d'incrémentation (`++`) et de décrémentation (`--`) de ce groupe sont des *opérateurs* de *préfixe*, c'est-à-dire qu'ils apparaissent avant l'opérande dans une expression. Les opérateurs de préfixe diffèrent de leur équivalent suffixe car l'opération d'incrémentation ou de décrémentation est effectuée avant le renvoi de la valeur de l'expression globale. Par exemple, le code suivant montre comment la valeur de l'expression `++xNum` est renvoyée après l'incrémentation de la valeur :

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

Tous les opérateurs unaires, comme répertoriés dans le tableau suivant, ont la même priorité :

Opérateur	Opération effectuée
`++`	Incrémentation (préfixe)
`--`	Décrémentation (préfixe)
`+`	Unaire +
`-`	Unaire - (négation)
`!`	NOT logique
`~`	NOT au niveau du bit
`delete`	Supprime une propriété
`typeof`	Renvoie les informations de type
`void`	Renvoie une valeur non définie

Opérateurs de multiplication

Les opérateurs de multiplication prennent deux opérandes et effectuent des multiplications, des divisions ou des calculs de modulo.

Tous les opérateurs de multiplication, comme répertoriés dans le tableau suivant, ont la même priorité :

Opérateur	Opération effectuée
`*`	Multiplication
`/`	Division
`%`	Modulo

Opérateurs d'ajout

Les opérateurs d'ajout prennent deux opérandes et effectuent des calculs d'addition ou de soustraction. Tous les opérateurs d'ajout, comme répertoriés dans le tableau suivant, ont la même priorité :

Opérateur	Opération effectuée
`+`	Addition
`-`	Soustraction

Opérateurs de décalage au niveau du bit

Ces opérateurs prennent deux opérandes et décalent les bits du premier selon la valeur spécifiée dans le second. Tous les opérateurs de décalage au niveau du bit, comme répertoriés dans le tableau suivant, ont la même priorité :

Opérateur	Opération effectuée
`<<`	Décalage gauche au niveau du bit
`>>`	Décalage droit au niveau du bit
`>>>`	Décalage droit non signé au niveau du bit

Opérateurs relationnels

Les opérateurs relationnels prennent deux opérandes, comparent leurs valeurs et renvoient une valeur booléenne. Tous les opérateurs relationnels, comme répertoriés dans le tableau suivant, ont la même priorité :

Opérateur	Opération effectuée
`<`	Inférieur à
`>`	Supérieur à
`<=`	Inférieur ou égal à
`>=`	Supérieur ou égal à
`as`	Vérifie le type de données
`in`	Vérifie les propriétés des objets
`instanceof`	Vérifie la chaîne de prototype
`is`	Vérifie le type de données

Opérateurs d'égalité

Les opérateurs d'égalité prennent deux opérandes, comparent leurs valeurs et renvoient une valeur booléenne. Tous les opérateurs d'égalité, comme répertoriés dans le tableau suivant, ont la même priorité :

Opérateur	Opération effectuée
`==`	Egalité
`!=`	Inégalité
`===`	Egalité stricte
`!==`	Inégalité stricte

Opérateurs logiques au niveau du bit

Ces opérateurs prennent deux opérandes et effectuent des opérations logiques au niveau des bits. La priorité de ces opérateurs diffère et ils sont présentés dans le tableau suivant par ordre décroissant de priorité :

Opérateur	Opération effectuée
`&`	AND au niveau du bit
`^`	XOR au niveau du bit
`|`	OR au niveau du bit

Opérateurs logiques

Les opérateurs logiques prennent deux opérandes et renvoient une valeur booléenne. La priorité de ces opérateurs diffère et ils sont présentés dans le tableau suivant par ordre décroissant de priorité :

Opérateur	Opération effectuée
`&&`	AND logique
`||`	OR logique

Opérateur conditionnel

L'opérateur conditionnel est un opérateur ternaire, c'est-à-dire qu'il prend trois opérandes. Il correspond à une méthode abrégée de l'application de l'instruction conditionnelle `if...else`.

Opérateur	Opération effectuée
`?:`	Conditionnel

Opérateurs d'affectation

Les opérateurs d'affectation prennent deux opérandes et affectent une valeur à l'un d'eux en fonction de la valeur de l'autre. Tous les opérateurs d'affectation, comme répertoriés dans le tableau suivant, ont la même priorité :

Opérateur	Opération effectuée
`=`	Affectation
`*=`	Affectation de multiplication
`/=`	Affectation de division
`%=`	Affectation modulo
`+=`	Affectation d'addition
`-=`	Affectation de soustraction
`<<=`	Affectation de décalage gauche au niveau du bit
`>>=`	Affectation de décalage droit au niveau du bit
`>>>=`	Affectation de décalage droit au niveau du bit non signé
`&=`	Affectation AND au niveau du bit
`^=`	Affectation XOR au niveau du bit
`|=`	Affectation OR au niveau du bit

Instructions conditionnelles

ActionScript 3.0 fournit trois instructions conditionnelles de base que vous pouvez utiliser pour contrôler le flux du programme.

if..else

L'instruction conditionnelle `if....else` permet de tester une condition, puis d'exécuter un bloc de code lorsque cette condition est positive et d'en exécuter un autre dans le cas contraire. Par exemple, le code suivant vérifie si la valeur de `x` est supérieure à 20 et génère une fonction `trace()` dans l'affirmative ou une autre fonction `trace()` dans le cas contraire :

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

Si vous ne souhaitez pas exécuter un autre bloc de code, vous pouvez utiliser l'instruction `if` sans l'instruction `else`.

if..else if

L'instruction conditionnelle `if...else if` permet de tester plusieurs conditions. Par exemple, le code suivant teste non seulement si la valeur de `x` est supérieure à 20, mais également si la valeur de `x` est négative:

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

Si une instruction `if` ou `else` est suivie d'une seule instruction, elle n'a pas besoin d'être placée entre accolades. Par exemple, le code suivant n'utilise pas d'accolades :

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

Néanmoins, Adobe conseille de toujours utiliser des accolades car un comportement inattendu peut se produire si des instructions sont ajoutées ultérieurement à une instruction conditionnelle sans accolades. Par exemple, dans le code suivant, la valeur de `positiveNums` augmente de 1 que la condition renvoie `true` ou non :

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

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

trace(positiveNums); // 1
```

switch

L'instruction `switch` est utile si vous avez plusieurs chemins d'exécution sur la même expression de condition. Elle s'apparente à une longue série d'instructions `if..else if`, mais est plus lisible. Au lieu de tester une condition pour une valeur booléenne, l'instruction `switch` évalue une expression et utilise le résultat pour déterminer le bloc de code à exécuter. Les blocs de code commencent par une instruction `case` et se terminent par une instruction `break`. Par exemple, l'instruction `switch` suivante imprime le jour de la semaine en fonction du numéro du jour renvoyé par la méthode `Date.getDay()` :

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

Boucle

Les instructions de bouclage vous permettent d'exécuter un bloc de code spécifique de façon répétée à l'aide d'une série de valeurs ou de variables. Adobe vous recommande de toujours placer le bloc de code entre accolades (`{}`). Vous pouvez omettre les accolades si le bloc de code contient une seule instruction mais, comme pour les instructions conditionnelles, cette pratique est déconseillée : il est en effet possible que des instructions ajoutées à une date ultérieure soient exclues du bloc de texte par inadvertance. Si vous ajoutez ultérieurement une instruction à inclure dans le bloc de code mais que vous oubliez d'ajouter les accolades nécessaires, l'instruction n'est pas exécutée dans la boucle.

for

La boucle `for` vous permet de faire une itération sur une variable pour une plage de valeurs spécifique. Vous devez indiquer trois expressions dans une instruction `for` : une variable définie sur une valeur initiale, une instruction conditionnelle qui détermine le moment où la boucle prend fin et une expression qui modifie la valeur de la variable avec chaque boucle. Par exemple, le code suivant boucle à cinq reprises. La valeur de la variable `i` commence à 0 et prend fin à 4, et le résultat est constitué par les nombres compris entre 0 et 4, chacun sur sa propre ligne.

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

for..in

La boucle `for....in` permet de faire une itération sur les propriétés d'un objet ou les éléments d'un tableau. Par exemple, utilisez la boucle `for....in` pour faire une itération sur les propriétés d'un objet générique (les propriétés d'un objet n'étant pas conservées dans un ordre particulier, elles peuvent apparaître dans un ordre imprévisible) :

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

Vous pouvez également faire une itération sur les éléments d'un tableau :

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

Vous ne pouvez pas itérer sur les propriétés d'un objet lorsqu'il s'agit d'une occurrence de classe définie par l'utilisateur, sauf si la classe est dynamique. Et même dans ce dernier cas, vous ne pouvez faire d'itération que sur les propriétés ajoutées dynamiquement.

for each...in

La boucle `for each....in` vous permet de faire une itération sur les éléments d'une collection (balises dans un objet XML ou XMLList, valeurs des propriétés d'un objet ou éléments d'un tableau). Ainsi, comme l'illustre l'extrait suivant, vous pouvez utiliser une boucle `for each...in` pour itérer sur les propriétés d'un objet générique mais, contrairement à la boucle `for....in`, la variable d'itération d'une boucle `for each...in` contient la valeur de la propriété plutôt que le nom de celle-ci :

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

Vous pouvez faire une itération sur un objet XML ou XMLList, comme l'indique l'exemple suivant :

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

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

Vous pouvez également faire une itération sur les éléments d'un tableau, comme l'indique cet exemple :

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

Vous ne pouvez pas faire une itération sur les propriétés d'un objet si ce dernier est une occurrence d'une classe scellée. Même pour les occurrences de classes dynamiques, vous ne pouvez pas faire une itération sur des propriétés fixes qui sont des propriétés définies comme faisant partie d'une définition de classe.

while

La boucle `while` est semblable à une instruction `if` qui se répète tant que la condition est `true`. Par exemple, le code suivant produit le même résultat que l'exemple de boucle `for` :

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

L'un des inconvénients que présente la boucle `while` par rapport à la boucle `for` est que les risques de boucles sans fin sont plus importants avec les boucles `while`. Par exemple, le code qui utilise la boucle `for` ne passe pas la compilation si vous omettez l'expression qui incrémente la variable du compteur, alors que le code qui utilise la boucle `while` est compilé. Et sans l'expression qui incrémente `i`, la boucle se poursuit sans fin.

do..while

La boucle `do..while` est une boucle `while` qui garantit que le bloc de code est exécuté au moins une fois, car la condition est vérifiée une fois que le bloc de code est exécuté. Le code suivant est un exemple simple d'une boucle `do..while` qui renvoie une sortie même si la condition n'est pas remplie.

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

Fonctions

Les *fonctions* sont des blocs de code qui effectuent des tâches spécifiques et qui peuvent être réutilisés dans votre programme. Il existe deux types de fonctions dans ActionScript 3.0 : les *méthodes* et les *fonctions closure*. Une fonction est appelée méthode ou fonction closure selon le contexte dans lequel elle est définie. Une fonction est appelée méthode si vous la définissez comme partie d'une définition de classe ou l'associez à l'occurrence d'un objet. Une fonction est appelée fonction closure si elle est définie de toute autre façon.

Les fonctions ont toujours été très importantes dans ActionScript. Dans ActionScript 1.0, par exemple, le mot-clé `class` n'existait pas. Par conséquent, les classes étaient définies par des fonctions constructeurs. Même si le mot-clé `class` a depuis été ajouté au langage, une solide compréhension des fonctions reste importante si vous souhaitez bénéficier de tous les avantages du langage. Ceci peut être un défi pour les programmeurs qui s'attendent à ce que les fonctions ActionScript se comportent de façon identique à celles des langages tels que C++ ou Java. Même si l'appel et la définition des fonctions de base ne devraient pas constituer un défi pour les programmeurs expérimentés, certaines des fonctions ActionScript les plus avancées nécessitent une explication.

Concepts des fonctions de base

Cette section décrit les techniques d'appel et de définition des fonctions de base.

Appel de fonctions

Vous appelez une fonction en utilisant son identifiant suivi de l'opérateur parenthèses (`()`). Vous placez les paramètres de fonction que vous souhaitez envoyer à la fonction entre parenthèses à l'aide de l'opérateur parenthèses. Par exemple, la fonction `trace()`, qui est une fonction de niveau supérieur dans ActionScript 3.0, est utilisée dans l'ensemble de ce manuel :

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

Si vous appelez une fonction sans paramètres, vous devez utiliser une paire de parenthèses vide. Par exemple, vous pouvez utiliser la méthode `Math.random()`, qui ne prend aucun paramètre, pour générer un nombre aléatoire :

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

Définition de vos fonctions

Dans ActionScript 3.0, deux techniques vous permettent de définir une fonction : vous pouvez utiliser une instruction de fonction ou une expression de fonction. La technique que vous choisissez dépend du style de programmation que vous préférez, plus statique ou dynamique. Définissez vos fonctions à l'aide d'instructions de fonction si vous préférez une programmation en mode strict, ou statique. Définissez vos fonctions à l'aide d'expressions de fonction si vous en avez vraiment besoin. Les expressions de fonction sont utilisées plus souvent dans la programmation en mode standard, ou dynamique.

Instructions de fonction

Les instructions de fonction représentent la technique privilégiée pour définir des fonctions en mode strict. Une instruction de fonction commence par le mot-clé `function`, suivi :

- du nom de la fonction ;
- des paramètres, dans une liste séparée par des virgules, placée entre parenthèses ;
- du corps de la fonction, c'est-à-dire le code ActionScript à exécuter lorsque la fonction est invoquée, placé entre accolades.

  Par exemple, le code suivant crée une fonction qui définit un paramètre puis appelle la fonction à l'aide de la chaîne “`hello"` comme valeur de paramètre :

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

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

Expressions de fonction

La deuxième façon de déclarer une fonction est d'utiliser une instruction d'affectation avec une expression de fonction, parfois appelée littéral de fonction ou fonction anonyme. Il s'agit d'une méthode plus détaillée largement utilisée dans les versions précédentes d'ActionScript.

Une instruction d'affectation associée à une expression de fonction commence par le mot-clé `var`, suivi :

- du nom de la fonction ;
- de l'opérateur deux points (`:`) ;
- de la classe `Function` pour indiquer le type de données ;
- de l'opérateur d'affectation (`=`) ;
- du mot-clé `function` ;
- des paramètres, dans une liste séparée par des virgules, placée entre parenthèses ;
- du corps de la fonction, c'est-à-dire le code ActionScript à exécuter lorsque la fonction est invoquée, placé entre accolades.

  Par exemple, le code suivant déclare la fonction `traceParameter` à l'aide d'une expression de fonction :

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

  Vous remarquerez que vous ne spécifiez pas de nom de fonction, comme dans une instruction de fonction . Une autre différence importante entre les expressions de fonction et les instructions de fonction est qu'une expression de fonction est plus une expression qu'une instruction. Ceci signifie qu'une expression de fonction ne peut pas être utilisée seule, contrairement à une instruction de fonction. Une expression de fonction peut être utilisée uniquement en tant que partie d'une instruction, généralement une instruction d'affectation. L'exemple suivant représente une expression de fonction affectée à un élément de tableau :

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

Choix d'instructions ou d'expressions

En règle générale, utilisez une instruction de fonction, à moins que des circonstances spécifiques exigent l'utilisation d'une expression. Les instructions de fonction sont moins détaillées et renforcent plus la cohérence entre le mode strict et le mode standard que les expressions de fonction.

Les instructions de fonction sont plus lisibles que les instructions d'affectation qui contiennent des expressions de fonction. Les instructions de fonction rendent votre code plus concis ; elles prêtent moins à confusion que les expressions de fonction, qui exigent l'utilisation des mots-clés `var` et `function`.

Les instructions de fonction renforcent la cohérence entre les deux modes de compilateur car vous pouvez utiliser la syntaxe à point en mode strict et en mode standard pour appeler une méthode déclarée à l'aide d'une instruction de fonction. Ceci ne s'applique pas nécessairement aux méthodes déclarées avec une expression de fonction. Par exemple, le code suivant définit la classe Example à l'aide de deux méthodes : `methodExpression()`, qui est déclarée par le biais d'une expression de fonction, et `methodStatement()`, qui est déclarée par le biais d'une instruction de fonction. En mode strict, vous ne pouvez pas utiliser la syntaxe à point pour invoquer la méthode `methodExpression()`.

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

var myEx:Example = new Example();
myEx.methodExpression(); // error in strict mode; okay in standard mode
myEx.methodStatement(); // okay in strict and standard modes
```

Les expressions de fonction sont plus adaptées à la programmation ciblée sur un comportement d'exécution ou dynamique. Si vous préférez utiliser le mode strict mais que vous souhaitez également appeler une méthode déclarée avec une expression de fonction, vous pouvez utiliser l'une des deux techniques. Premièrement, vous pouvez appeler la méthode à l'aide de l'opérateur crochets (`[]`) au lieu de l'opérateur point (`.`). L'appel de méthode suivant a lieu à la fois en mode strict et en mode standard :

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

Deuxièmement, vous pouvez déclarer la classe entière comme classe dynamique. Même si ceci vous permet d'appeler la méthode à l'aide de l'opérateur point, l'inconvénient est que vous sacrifiez une fonctionnalité de mode strict pour toutes les occurrences de cette classe. Par exemple, le compilateur ne génère pas d'erreur si vous tentez d'accéder à une propriété non définie sur une occurrence d'une classe dynamique.

Les expressions de fonction peuvent être utiles dans certains cas. Elles sont couramment utilisées pour des fonctions qui sont utilisées une seule fois. Elles peuvent être utilisées également pour associer une fonction à une propriété de prototype. Pour plus d'informations, consultez la section « Objet prototype » à la page 123.

Il existe deux légères différences entre les instructions de fonction et les expressions de fonction dont il faut tenir compte lorsque vous choisissez la technique à utiliser. La première réside dans le fait que les expressions de fonction n'existent pas indépendamment en tant qu'objets en ce qui concerne la gestion de la mémoire et le nettoyage. En d'autres termes, lorsque vous affectez une expression de fonction à un autre objet (un élément de tableau ou une propriété d'objet, par exemple) vous créez l'unique référence à cette expression de fonction dans votre code. Si le tableau ou l'objet auquel est associée l'expression de fonction n'est plus disponible, vous n'avez plus accès à l'expression de fonction. Si le tableau ou l'objet est supprimé, la mémoire que l'expression de fonction utilise peut être nettoyée, ce qui signifie que la mémoire peut être réutilisée à d'autres fins.

L'exemple suivant indique que pour une expression de fonction, la fonction n'est plus disponible une fois que la propriété à laquelle l'expression est affectée est supprimée. La classe Test est dynamique, ce qui signifie que vous pouvez ajouter une propriété appelée `functionExp` qui contient une expression de fonction. La fonction `functionExp()` peut être appelée avec l'opérateur point, mais une fois que la propriété `functionExp` est supprimée, la fonction n'est plus accessible.

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

// function expression
myTest.functionExp = function () { trace("Function expression") };
myTest.functionExp(); // Function expression
delete myTest.functionExp;
myTest.functionExp(); // error
```

Si, en revanche, la fonction est d'abord définie avec une instruction de fonction, elle existe comme son propre objet et continue à exister, même une fois que vous avez supprimé la propriété à laquelle elle est associée. L'opérateur `delete` fonctionne uniquement sur les propriétés d'objets, donc même un appel à supprimer la fonction `stateFunc()` ne fonctionne pas.

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

// function statement
function stateFunc() { trace("Function statement") }
myTest.statement = stateFunc;
myTest.statement(); // Function statement
delete myTest.statement;
delete stateFunc; // no effect
stateFunc();// Function statement
myTest.statement(); // error
```

La deuxième différence entre les instructions de fonction et les expressions de fonction réside dans le fait que les instructions de fonction existent dans le cadre dans lequel elles sont définies, notamment les instructions qui apparaissent avant l'instruction de fonction. Les expressions de fonction, en revanche, sont définies uniquement pour les instructions ultérieures. Par exemple, le code suivant appelle la fonction `scopeTest()` avant qu'elle soit définie :

```
statementTest(); // statementTest

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

Les expressions de fonction ne sont pas disponibles avant d'être définies. Par conséquent, le code suivant provoque une erreur d'exécution :

```
expressionTest(); // run-time error

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

Renvoi de valeurs des fonctions

Pour renvoyer une valeur de votre fonction, utilisez l'instruction `return` suivie de l'expression ou de la valeur littérale que vous souhaitez renvoyer. Par exemple, le code suivant renvoie une expression représentant le paramètre :

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

Vous remarquerez que l'instruction `return` termine la fonction, de sorte que les instructions suivant une instruction `return` ne sont pas exécutées, comme suit :

```
function doubleNum(baseNum:int):int {
    return (baseNum * 2);
    trace("after return"); // This trace statement will not be executed.
}
```

En mode strict, vous devez renvoyer une valeur du type approprié si vous choisissez de spécifier un type de renvoi. Par exemple, le code suivant génère une erreur en mode strict car il ne renvoie pas de valeur valide :

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

Fonctions imbriquées

Vous pouvez imbriquer des fonctions, ce qui signifie que vous pouvez déclarer des fonctions au sein d'autres fonctions. Une fonction imbriquée est disponible uniquement dans sa fonction parent, à moins qu'une référence à la fonction soit transmise à un code externe. Par exemple, le code suivant déclare deux fonctions imbriquées à l'intérieur de la fonction `getNameAndVersion()` :

```
function getNameAndVersion():String
{
    function getVersion():String
    {
        return "10";
    }
    function getProductName():String
    {
        return "Flash Player";
    }
    return (getProductName() + " " + getVersion());
}
trace(getNameAndVersion()); // Flash Player 10
```

Lorsque des fonctions imbriquées sont transmises à un code externe, elles le sont en tant que fonctions closure, ce qui signifie que la fonction conserve les définitions se trouvant dans le domaine au moment de la définition de la fonction. Pour plus d'informations, consultez la section « Domaine d'une fonction » à la page 91.

Paramètres de fonction

ActionScript 3.0 permet d'exploiter des paramètres de fonction qui peuvent sembler nouveaux pour les programmeurs qui découvrent le langage. Bien que la plupart des programmeurs connaissent le principe de transfert de paramètres par valeur ou référence, l'objet `arguments` et le paramètre ... (rest) seront peut-être des nouveautés.

Transfert d'arguments par valeur ou par référence

Dans de nombreux langages de programmation, il est important de comprendre la différence entre le transfert d'arguments par valeur ou par référence car elle peut affecter la façon dont le code est conçu.

Transférer par valeur signifie que la valeur de l'argument est copiée dans une variable locale pour être utilisée dans la fonction. Transférer par référence signifie que seule une référence à l'argument est transmise, au lieu de la valeur réelle. Aucune copie de l'argument réel n'est effectuée. A la place, une référence à la variable transférée en tant qu'argument est créée et affectée à une variable locale pour être utilisée dans la fonction. En tant que référence à une variable en dehors de la fonction, la variable locale vous permet de modifier la valeur de la variable d'origine.

Dans ActionScript 3.0, tous les arguments sont transférés par référence car toutes les valeurs sont stockées en tant qu'objets. Néanmoins, les objets qui appartiennent aux types de données primitifs (Boolean, Number, int, uint et String) possèdent des opérateurs spéciaux qui font qu'ils se comportent comme s'ils étaient transférés par valeur. Par exemple, le code suivant crée une fonction appelée `passPrimitives()` qui définit deux paramètres appelés `xParam` et `yParam` de type int. Ces paramètres sont identiques aux variables locales déclarées dans le corps de la fonction `passPrimitives()`. Lorsque la fonction est appelée avec les arguments `xValue` et `yValue`, les paramètres `xParam` et `yParam` sont initialisés avec des références aux objets int représentés par `xValue` et `yValue`. Les arguments se comportent comme s'ils étaient transférés par valeur car ils sont primitifs. Bien que `xParam` et `yParam` contiennent initialement des références aux objets `xValue` et `yValue` uniquement, toute modification apportée aux variables dans le corps de fonction génère de nouvelles copies des valeurs dans la mémoire.

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

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

Dans la fonction `passPrimitives()`, les valeurs de `xParam` et `yParam` sont incrémentées, mais ceci n'affecte pas les valeurs de `xValue` et `yValue`, comme indiqué dans la dernière instruction `trace` . Ceci s'applique même si les paramètres portent les mêmes noms que les variables, `xValue` et `yValue`, car les `xValue` et `yValue` se trouvant dans la fonction pointeraient vers de nouveaux emplacements dans la mémoire qui existent indépendamment des variables du même nom en dehors de la fonction.

Tous les autres objets (c'est-à-dire les objets qui n'appartiennent pas aux types de données primitifs) sont toujours transférés par référence, ce qui vous permet de modifier la valeur de la variable d'origine. Par exemple, le code suivant crée un objet appelé `objVar` avec deux propriétés, `x` et `y`. L'objet est transféré en tant qu'argument à la fonction `passByRef()`. Etant donné que l'objet n'est pas un type primitif, non seulement il est transféré par référence mais il reste également une référence. Ceci signifie que les changements effectués sur les paramètres dans la fonction affectent les propriétés d'objet en dehors de la fonction.

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

Le paramètre `objParam` référence le même objet que la variable globale `objVar`. Comme vous pouvez le constater dans les instructions `trace` de l'exemple, les modifications apportées aux propriétés `x` et `y` de l'objet `objParam` sont visibles dans l'objet `objVar`.

Valeurs de paramètre par défaut

Dans ActionScript 3.0, il est à présent possible de déclarer des *valeurs de paramètre par défaut* pour une fonction. Si un appel à une fonction avec des valeurs de paramètre par défaut omet un paramètre avec des valeurs par défaut, la valeur spécifiée dans la définition de fonction pour ce paramètre est utilisée. Tous les paramètres avec des valeurs par défaut doivent être placés à la fin de la liste des paramètres. Les valeurs affectées comme valeurs par défaut doivent être des constantes de compilation. L'existence d'une valeur par défaut pour un paramètre le rend *facultatif*. Un paramètre sans valeur par défaut est considéré comme un *paramètre obligatoire*.

Par exemple, le code suivant crée une fonction avec trois paramètres, dont deux possèdent des valeurs par défaut . Lorsque la fonction est appelée avec un seul paramètre, les valeurs par défaut des paramètres sont utilisées.

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

Objet arguments

Lorsque les paramètres sont transférés à une fonction, vous pouvez utiliser l'objet `arguments` pour accéder aux informations concernant les paramètres transférés à votre fonction. Voici certains aspects importants de l'objet `arguments` :

- L'objet `arguments` est un tableau qui comprend tous les paramètres transférés à la fonction.
- La propriété `arguments.length` indique le nombre de paramètres transmis à la fonction.
- La propriété `arguments.callee` fournit une référence à la fonction elle-même, ce qui est utile pour les appels récursifs à des expressions de fonction.

  Remarque : *l'objet `arguments` n'est pas disponible si un paramètre est appelé `arguments` ou si vous utilisez le paramètre ... (rest).*

  Si l'objet `arguments` est référencé dans le corps d'une fonction, ActionScript 3.0 permet aux appels de fonction d'inclure plus de paramètres que ceux définis dans la définition de fonction. Mais il génère une erreur de compilateur en mode strict si le nombre de paramètres ne correspond pas au nombre de paramètres obligatoires (et, éventuellement, au nombre de paramètres facultatifs). Vous pouvez utiliser l'aspect de tableau de l'objet `arguments` pour accéder aux paramètres transférés à la fonction, que ces paramètres soient définis ou non dans la définition de fonction. L'exemple suivant, qui est uniquement compilé en mode standard, utilise le tableau `arguments` et la propriété `arguments.length` pour suivre tous les paramètres transférés à la fonction `traceArgArray()` :

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

traceArgArray(1, 2, 3);

// output:
// 1
// 2
// 3
```

La propriété `arguments.callee` est souvent utilisée dans des fonctions anonymes pour créer une récursivité. Vous pouvez l'utiliser pour ajouter de la flexibilité à votre code. Si le nom de la fonction récursive change pendant votre cycle de développement, il est inutile de modifier l'appel récursif dans le corps de votre fonction si vous utilisez `arguments.callee` au lieu du nom de fonction. La propriété `arguments.callee` est utilisée dans l'expression de fonction suivante pour activer la récursivité :

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

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

Si vous utilisez le paramètre ... (rest) dans votre déclaration de fonction, l'objet `arguments` n'est pas disponible. Vous devez accéder aux paramètres à l'aide des noms de paramètre que vous avez déclarés.

N'utilisez pas la chaîne `"arguments"` comme nom de paramètre car elle masque l'objet `arguments`. Par exemple, si la fonction `traceArgArray()` est réécrite de façon à ce qu'un paramètre `arguments` soit ajouté, les références à `arguments` dans le corps de la fonction se réfèrent au paramètre plutôt qu'à l'objet `arguments`. Le code suivant donne le résultat :

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

traceArgArray(1, 2, 3);

// no output
```

L'objet `arguments` des versions précédentes d'ActionScript contenait également une propriété appelée `caller`, qui est une référence à la fonction qui appelait la fonction actuelle. La propriété `caller` n'existe pas dans ActionScript 3.0, mais si vous avez besoin d'une référence à la fonction d'appel, vous pouvez modifier celle-ci de façon à ce qu'elle transfère un paramètre supplémentaire qui en soit une référence.

Paramètre ... (rest)

ActionScript 3.0 présente une nouvelle déclaration de paramètre appelée le paramètre .. (rest). Ce paramètre vous permet de spécifier un paramètre de tableau qui accepte n'importe quel nombre d'arguments séparés par des virgules. Veillez à ne pas inclure un mot réservé dans le nom du paramètre. Cette déclaration de paramètre doit être le dernier paramètre spécifié. Ce paramètre rend l'objet `arguments` non disponible. Bien que le paramètre ... (rest) offre la même fonctionnalité que le tableau `arguments` et la propriété `arguments.length`, il ne fournit pas la même fonctionnalité que la propriété `arguments.callee`. Vérifiez que vous n'avez pas besoin d'utiliser `arguments.callee` avant d'utiliser le paramètre ... (rest).

L'exemple suivant réécrit la fonction `traceArgArray()` à l'aide du paramètre ... (rest) plutôt que de l'objet `arguments` :

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

traceArgArray(1, 2, 3);

// output:
// 1
// 2
// 3
```

Le paramètre ... (rest) peut également être utilisé avec d'autres paramètres, sous réserve qu'il soit le dernier de la liste. L'exemple suivant modifie la fonction `traceArgArray()` de façon à ce que son premier paramètre, `x`, soit de type int, et que le second utilise le paramètre ... (rest). Le résultat ignore la première valeur car le premier paramètre ne fait plus partie du tableau créé par le paramètre ... (rest).

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

traceArgArray(1, 2, 3);

// output:
// 2
// 3
```

Fonctions comme objets

Dans ActionScript 3.0, les fonctions sont des objets. Lorsque vous créez une fonction, vous créez un objet qui peut non seulement être transféré en tant que paramètre à une autre fonction, mais auquel sont également associées des propriétés et des méthodes.

Les fonctions transférées en tant qu'arguments à une autre fonction sont transmises par référence et non par valeur. Lorsque vous transférez une fonction en tant qu'argument, vous utilisez uniquement l'identifiant et non l'opérateur parenthèses qui permet d'appeler la méthode. Par exemple, le code suivant transfère une fonction appelée `clickListener()` en tant qu'argument à la méthode `addEventListener()` :

```
addEventListener(MouseEvent.CLICK, clickListener);
```

La méthode `Array.sort()` définit également un paramètre qui accepte une fonction. Pour consulter un exemple de fonction de tri personnalisée transférée en tant qu'argument à la fonction `Array.sort()`, consultez la section « Tri d'un tableau » à la page 166

Même si cela peut sembler étrange pour les programmeurs découvrant ActionScript, les fonctions peuvent avoir des propriétés et des méthodes, comme les objets. Chaque fonction a en réalité une propriété en lecture seule appelée `length` qui stocke le nombre de paramètres définis pour la fonction. Ceci est différent de la propriété `arguments.length` qui indique le nombre d'arguments envoyés à la fonction. Dans ActionScript, le nombre d'arguments envoyés à une fonction peut dépasser le nombre de paramètres définis pour cette dernière. L'exemple suivant (qui compile uniquement en mode standard car le mode strict exige une correspondance exacte entre le nombre d'arguments transférés et le nombre de paramètres définis) illustre la différence entre les deux propriétés :

```
// Compiles only in standard mode
function traceLength(x:uint, y:uint):void
{
    trace("arguments received: " + arguments.length);
    trace("arguments expected: " + traceLength.length);
}

traceLength(3, 5, 7, 11);
/* output:
arguments received: 4
arguments expected: 2 */
```

En mode standard, vous pouvez définir vos propriétés en dehors du corps de votre fonction. Les propriétés de fonction peuvent servir de propriétés quasi statiques vous permettant de sauvegarder l'état d'une variable liée à la fonction. Par exemple, vous pouvez suivre le nombre d'appels d'une fonction particulière. Une telle fonctionnalité peut être utile si vous écrivez un jeu et souhaitez suivre le nombre de fois qu'un utilisateur se sert d'une certaine commande (vous pouvez également utiliser une propriété de classe statique). L'exemple suivant (qui compile uniquement en mode standard car le mode strict n'autorise pas l'ajout de propriétés dynamiques aux fonctions) crée une propriété de fonction en dehors de la déclaration de la fonction et incrémente cette propriété à chaque appel de la fonction :

```
// Compiles only in standard mode
var someFunction:Function = function ():void
{
    someFunction.counter++;
}

someFunction.counter = 0;

someFunction();
someFunction();
trace(someFunction.counter); // 2
```

Domaine d'une fonction

Le domaine d'une fonction détermine non seulement l'endroit où cette fonction peut être appelée dans un programme, mais également les définitions auxquelles la fonction peut accéder. Les mêmes règles de domaine qui s'appliquent aux identifiants de variable s'appliquent aux identifiants de fonction. Une fonction déclarée dans le domaine global est disponible dans tout votre code. Par exemple, ActionScript 3.0 contient des fonctions globales (`isNaN()` et `parseInt()`, par exemple) disponibles n'importe où dans votre code. Une fonction imbriquée (une fonction déclarée dans une autre fonction) peut être utilisée n'importe où dans la fonction dans laquelle elle a été déclarée.

Chaîne de domaine

Chaque fois qu'une fonction commence une exécution, des objets et des propriétés sont créés. Premièrement, un objet spécial appelé *objet d'activation* est créé. Il stocke les paramètres et les variables locales ou fonctions déclarées dans le corps de la fonction. Vous ne pouvez pas accéder directement à l'objet d'activation car il s'agit d'un mécanisme interne. Deuxièmement, une *chaîne de domaine* est créée. Elle contient une liste ordonnée d'objets dans laquelle Flash Player ou Adobe AIR recherche des déclarations d'identifiant. Chaque fonction qui s'exécute a une chaîne de domaine stockée dans une propriété interne. Dans le cas d'une fonction imbriquée, la chaîne de domaine commence avec son objet d'activation, suivi par l'objet d'activation de sa fonction parent. La chaîne continue de cette façon jusqu'à ce que l'objet global soit atteint. L'objet global est créé lorsqu'un programme ActionScript commence, et contient toutes les fonctions et les variables globales.

Fonctions closure

Une fonction *closure* est un objet qui contient un instantané d'une fonction et de son *environnement lexical*. L'environnement lexical d'une fonction comprend toutes les variables, propriétés, méthodes et les objets dans la chaîne de domaine de la fonction, ainsi que leurs valeurs. Les fonctions closure sont créées chaque fois qu'une fonction est exécutée à part d'un objet ou d'une classe. Le fait que les fonctions closure conservent le domaine dans lequel elles ont été définies crée des résultats intéressants lorsqu'une fonction est transférée en tant qu'argument ou valeur de renvoi dans un domaine différent.

Par exemple, le code suivant crée deux fonctions : `foo()`, qui renvoie une fonction imbriquée appelée `rectArea()` qui calcule la surface d'un rectangle, et `bar()`, qui appelle `foo()` et stocke la fonction closure renvoyée dans une variable appelée `myProduct`. Même si la fonction `bar()` définit sa propre variable locale `x` (avec une valeur de 2), lorsque la fonction closure `myProduct()` est appelée, elle conserve la variable `x` (avec une valeur de 40) définie dans la fonction `foo()`. La fonction `bar()` renvoie par conséquent la valeur `160` au lieu de `8`.

```
function foo():Function
{
    var x:int = 40;
    function rectArea(y:int):int // function closure defined
    {
        return x * y
    }
    return rectArea;
}
function bar():void
{
    var x:int = 2;
    var y:int = 4;
    var myProduct:Function = foo();
    trace(myProduct(4)); // function closure called
}
bar(); // 160
```

Les méthodes se comportent de la même façon car elles conservent également les informations concernant l'environnement lexical dans lequel elles ont été créées. Cette caractéristique se remarque plus particulièrement lorsqu'une méthode est extraite de son occurrence, ce qui crée une méthode liée. La différence principale entre une fonction closure et une méthode liée est que la valeur du mot-clé `this` dans une méthode liée se réfère toujours à l'occurrence à laquelle elle était associée à l'origine, alors que dans une fonction closure, la valeur du mot-clé `this` peut changer. Pour plus d'informations, consultez la section « Méthodes » à la page 101.

Chapitre 5 : Programmation orientée objets en ActionScript

Ce chapitre décrit les éléments d'ActionScript qui prennent en charge la programmation orientée objets (POO). Il ne décrit pas les principes généraux de la POO, tels que la conception des objets, l'abstraction, l'encapsulation, l'héritage et le polymorphisme. Ce chapitre se concentre sur l'application de ces principes en ActionScript 3.0.

Dans la mesure où ActionScript découle d'un langage de script, la prise en charge de la programmation orientée objets est facultative en ActionScript 3.0. Les programmeurs peuvent ainsi choisir l'approche qui leur convient le mieux, en fonction de la portée et de la complexité de chaque projet. Pour les petites tâches, il peut être préférable d'utiliser ActionScript suivant le paradigme de la programmation par procédures. Pour les projets plus importants, l'application des principes de la POO peut rendre votre code plus facile à comprendre, à maintenir et à développer.

Principes de base de la programmation orientée objets

Introduction à la programmation orientée objets

La programmation orientée objets (POO) est une technique d'organisation du code d'un programme en le groupant en objets, les objets étant ici des éléments individuels comportant des informations (valeurs de données) et des fonctionnalités. L'approche orientée objet permet de regrouper des éléments particuliers d'informations (par exemple, les informations d'un enregistrement musical: titre de l'album, titre de la piste ou nom de l'artiste) avec des fonctionnalités ou des actions communes associées à ces informations (comme l'ajout de la piste à une liste de lecture ou bien la lecture de tous les enregistrement de cet artiste). Ces éléments sont rassemblés en un seul, l'objet (par exemple, un « Album ».ou une « piste »). La possibilité d'intégrer ainsi toutes ces valeurs et ces fonctions offre divers avantages : par exemple, il est possible de ne suivre qu'une seule variable plutôt que plusieurs d'entre elles, de regrouper des fonctionnalités liées entre elles et de structurer les programmes pour qu'ils se rapprochent davantage du fonctionnement humain.

Tâches courantes de la programmation orientée objets

Dans la pratique, la programmation orientée objets se décompose en deux parties. La première est l'ensemble des stratégies et des techniques de conception d'un programme (souvent appelée *conception orientée objets*). Il s'agit là d'un vaste sujet qui ne sera pas abordé dans ce chapitre. L'autre partie de la programmation orientée objets est l'ensemble des structures de programmation qui sont disponibles dans un langage donné pour faciliter la construction d'un programme selon une approche orientée objets. Ce chapitre aborde les tâches suivantes, qui sont fréquentes en POO :

- Définition des classes
- Création de propriétés, méthodes et accesseurs get et set (méthodes accesseurs)
- Contrôle de l'accès aux classes, propriétés, méthodes et accesseurs
- Création de propriétés et de méthodes statiques
- Création de structures d'énumération
- Définition et utilisation d'interfaces
- Utilisation de l'héritage, y compris lors de la redéfinition des éléments des classes

Concepts et termes importants

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Attribut : caractéristique affectée à un élément de classe (comme une propriété ou une méthode) dans la définition de cette classe. Les attributs sont couramment utilisés pour définir le niveau d'accès à cette propriété ou cette méthode par du code situé dans d'autres parties du programme. Par exemple, `private` et `public` sont des attributs. Une méthode privée ne peut être appelée que par le code résidant dans sa classe, alors qu'une méthode publique peut être appelée par du code résidant n'importe où dans le programme.
- Classe : définition de la structure et du comportement des objets d'un certain type (comme un modèle ou une matrice pour des objets de ce type de données).
- Hiérarchie des classes : structure de plusieurs classes apparentées, qui indique quelles classes héritent de fonctionnalités des autres classes.
- Constructeur : méthode spéciale pouvant être définie dans une classe et qui est appelée lors de la création d'une occurrence de cette classe. Un constructeur est couramment utilisé pour spécifier des valeurs par défaut ou effectuer des opérations de configuration pour l'objet.
- Type de données : type d'informations qui peut être stocké dans une variable particulière. En règle générale, *type de données* a la même signification que *classe*.
- Opérateur point : en ActionScript et dans de nombreux autres langages de programmation, le signe point (`.`) indique qu'un nom fait référence à un élément enfant (propriété ou méthode) d'un objet. Par exemple, dans l'expression `myObject.myProperty`, l'opérateur point indique que le terme `myProperty` se réfère à une valeur qui est un élément de l'objet appelé `myObject`.
- Enumération : ensemble de valeurs constantes apparentées, regroupées pour des raisons pratiques sous forme de propriétés d'une même classe.
- Héritage : mécanisme de la programmation orientée objets qui permet à la définition d'une classe de comporter toutes les fonctionnalités de la définition d'une autre classe (en général, en y ajoutant de nouvelles fonctionnalités).
- occurrence : objet réel créé dans le cadre d'un programme.
- Espace de nom : essentiellement, attribut personnalisé qui autorise plus de contrôle pour définir le code qui est autorisé à accéder à un autre code.

Utilisation des exemples figurant dans les chapitres

Au fur et à mesure que vous avancez dans le chapitre, vous pouvez tester des exemples de code. Etant donné que les exemples de code de ce chapitre traitent principalement de la définition et de la manipulation des types de données, le test des exemples impliquera la création d'une occurrence de la classe définie, la manipulation de cette occurrence à l'aide de ses propriétés ou de ses méthodes et l'affichage des valeurs des propriétés de cette occurrence. Pour afficher ces valeurs, écrivez des valeurs dans une occurrence de champ de texte sur la scène ou bien utilisez la fonction `trace()` pour imprimer des valeurs sur le panneau Sortie. Ces techniques sont décrites en détail dans la section « Test des exemples de code contenus dans un chapitre » à la page 36.

Classes

Une classe est une représentation abstraite d'un objet. Une classe conserve des informations sur les types de données contenues par un objet et sur les comportements possibles de cet objet. L'utilité de ce niveau d'abstraction peut ne pas être évidente dans le cas de petits scripts ne contenant que quelques objets destinés à interagir les uns avec les autres. Cependant, au fur et à mesure qu'un programme croît en ampleur et que le nombre d'objets à gérer augmente, vous découvrirez probablement que les classes autorisent un meilleur contrôle sur la création des objets et sur leurs interactions.

Dès la première version d'ActionScript, les programmeurs en ActionScript pouvaient utiliser des objets Function pour créer des éléments ressemblant à des classes. ActionScript 2.0 a ensuite ajouté une prise en charge formelle des classes, avec des mots-clés tels que `class` et `extends`. De son côté, ActionScript 3.0 préserve la prise en charge des mots-clés introduits avec ActionScript 2.0, tout en ajoutant de nouvelles possibilités : par exemple un meilleur contrôle d'accès avec les attributs `protected` et `internal` ainsi qu'un meilleur contrôle de l'héritage avec les mots-clés `final` et `override`.

Si vous avez déjà créé des classes dans des langages de programmation tels que Java, C++ ou C#, vous ne serez pas dépaysés par ActionScript. ActionScript partage avec ces langages de nombreux mots-clés et noms d'attributs, comme `class`, `extends` et `public` qui seront abordés dans les sections suivantes.

Remarque : *dans ce chapitre, le terme propriété désigne tout membre d'un objet ou d'une classe (variables, constantes et méthodes). En outre, bien que les termes classe et statique soient fréquemment utilisés indifféremment, nous ferons une distinction entre ces termes dans ce chapitre. Par exemple, dans ce chapitre, l'expression propriétés de classe désigne tous les membres d'une classe plutôt que ses membres statiques exclusivement.*

Définitions de classe

En ActionScript 3.0, les définitions de classe utilisent la même syntaxe qu'en ActionScript 2.0. La syntaxe correcte d'une définition de classe utilise le mot-clé `class` suivi du nom de la classe. Le corps de la définition de classe est inséré entre des accolades (`{}`) après le nom de la classe. Par exemple, le code suivant crée une classe appelée Shape qui contient une variable appelée `visible` :

```
public class Shape
{
    var visible:Boolean = true;
}
```

Notez que la syntaxe est différente dans le cas des définitions de classe faisant partie d'un package. En ActionScript 2.0, si une classe fait partie d'un package, le nom de ce dernier doit figurer dans la déclaration de classe. Comme l'instruction `package` a été introduite en ActionScript 3.0, le nom du package doit figurer dans la déclaration de package et non pas dans la déclaration de classe. Par exemple, les déclarations de classe suivantes montrent comment la classe BitmapData, qui fait partie du package flash.display, est définie dans ActionScript 2.0 et ActionScript 3.0 :

```
// ActionScript 2.0
class flash.display.BitmapData {}

// ActionScript 3.0
package flash.display
{
    public class BitmapData {}
}
```

Attributs de classe

ActionScript 3.0 vous permet de modifier des définitions de classe à l'aide de l'un des quatre attributs suivants :

Attribut	Définition
`dynamique`	Permet d'ajouter des propriétés aux occurrences lors de l'exécution.
`final`	Ne doit pas être étendue par une autre classe.
`interne` (par défaut)	Visible pour les références à partir du package actuel.
`public`	Visible pour les références à partir de n'importe quel point du code.

Pour chacun de ces attributs, à l'exception d'`internal`, vous devez inclure explicitement l'attribut pour obtenir le comportement qui lui est associé. Par exemple, faute d'inclure l'attribut `dynamic` lors de la définition d'une classe, vous ne serez pas en mesure d'ajouter des propriétés à une occurrence de classe au cours de son exécution. Pour affecter explicitement un attribut, placez-le au début de la définition de la classe, comme dans le code ci-dessous :

```
dynamic class Shape {}
```

Vous pouvez remarquer que la liste ne contient pas d'attribut appelé `abstract`. En effet, les classes abstraites ne sont pas prises en charge en ActionScript 3.0. Vous pouvez aussi remarquer que la liste ne comprend pas non plus les attributs `private` et `protected`. Ces attributs n'ont de signification qu'au sein d'une définition de classe et ne peuvent être appliqués aux classes elles-mêmes. Si vous ne souhaitez pas qu'une classe soit visible hors de son package, placez-la au sein d'un package et affectez la classe de l'attribut `internal`. Autrement, vous pouvez omettre les attributs `internal` et `public` et l'ordinateur ajoutera automatiquement l'attribut `internal` pour vous. Si vous ne souhaitez pas qu'une classe soit visible à l'extérieur du fichier source dans lequel elle est définie, placez-la à la fin de ce fichier source, après l'accolade de fin de la définition du package.

Corps de la classe

Le corps de la classe, qui est entouré d'accolades, est utilisé pour définir les variables, les constantes et les méthodes de la classe. L'exemple suivant montre la déclaration de la classe Accessibility dans l'API d'Adobe Flash Player :

```
public final class Accessibility
{
    public static function get active():Boolean;
    public static function updateProperties():void;
}
```

Vous pouvez aussi définir un espace de noms au sein d'un corps de classe. L'exemple suivant montre la définition d'un espace de noms dans le corps d'une classe et son utilisation comme attribut d'une méthode de cette classe :

```
public class SampleClass
{
    public namespace sampleNamespace;
    sampleNamespace function doSomething():void;
}
```

ActionScript 3.0 vous permet d'inclure dans un corps de classe non seulement des définitions, mais également des instructions. Les instructions qui figurent dans le corps d'une classe, mais hors d'une définition de méthode, sont exécutées une seule fois, lors de la première apparition de la définition de classe et de la création de l'objet class qui lui est associé. L'exemple suivant comprend un appel vers une fonction externe, `hello()` et une instruction `trace` qui affiche un message de confirmation lorsque la classe est définie.

```
function hello():String
{
    trace("hola");
}
class SampleClass
{
    hello();
    trace("class created");
}
// output when class is created
hola
class created
```

Contrairement aux versions antérieures d'ActionScript, en ActionScript 3.0 il est permis de définir une propriété static et une propriété d'occurrence ayant le même nom dans le corps de la même classe. Par exemple, le code suivant déclare une variable statique appelée `message` et une variable d'occurrence du même nom :

```
class StaticTest
{
    static var message:String = "static variable";
    var message:String = "instance variable";
}
// In your script
var myST:StaticTest = new StaticTest();
trace(StaticTest.message); // output: static variable
trace(myST.message); // output: instance variable
```

Attributs de propriété de classe

Dans le cadre du modèle d'objet ActionScript, le terme *propriété* représente tout ce qui peut être membre d'une classe : variables, constantes et méthodes. Ce terme est utilisé de manière plus restrictive dans le manuel Références et composants du langage ActionScript 3.0 : il ne désigne alors que les membres d'une classe qui sont des variables ou qui sont définis par une méthode de lecture/définition. En ActionScript 3.0, il existe un jeu d'attributs qui peut être utilisé avec n'importe quelle propriété de classe. Le tableau suivant présente ce jeu d'attributs.

Attribut	Définition
`internal` (par défaut)	Visible pour les références au sein d'un même package.
`private`	Visible pour les références au sein d'une même classe.
`protected`	Visible pour des références au sein d'une même classe et de classes dérivées.
`public`	Visible pour des références en tous lieux.
`static`	Spécifie qu'une propriété appartient à la classe et non pas aux occurrences de la classe.
`UserDefinedNamespace`	Nom d'espace de noms défini par l'utilisateur.

Attributs d'espace de noms pour le contrôle d'accès

ActionScript 3.0 comporte quatre attributs spéciaux qui contrôlent l'accès aux propriétés définies au sein d'une classe : `public`, `private`, `protected` et `internal`.

Avec l'attribut `public`, une propriété est visible de n'importe quel point du script. Par exemple, si vous souhaitez qu'une méthode soit disponible pour du code externe au package, vous devez la déclarer avec l'attribut `public`. Ceci est valable pour toute propriété, qu'elle soit déclarée à l'aide des mots-clés `var`, `const` ou `function`.

Avec l'attribut `private`, une propriété n'est visible qu'à partir de la classe où cette propriété est définie. Ce comportement est différent de celui de l'attribut `privé` en ActionScript 2.0, où une sous-classe pouvait accéder à une propriété déclarée private d'une superclasse. L'accès en cours d'exécution constitue aussi un changement radical de comportement. En ActionScript 2.0, la restriction d'accès introduite par le mot-clé `private` ne portait que sur la compilation et il était facile de la contourner lors de l'exécution. Cette situation n'existe plus en ActionScript 3.0. Les propriétés désignées comme `private` sont indisponibles, aussi bien au cours de la compilation que de l'exécution.

Par exemple, le code ci-dessous crée une classe simple appelée PrivateExample pourvue d'une variable avec l'attribut private. Elle tente ensuite d'accéder à cette variable depuis un emplacement hors de la classe. En ActionScript 2.0, l'accès à cette variable en cours de compilation n'était pas possible. Mais il était facile de contourner cette restriction à l'aide de l'opérateur d'accès à la propriété (`[]`) qui se charge de la recherche des propriétés à l'exécution plutôt qu'à la compilation.

```
class PrivateExample
{
    private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);// compile-time error in strict mode
trace(myExample["privVar"]); // ActionScript 2.0 allows access, but in ActionScript 3.0, this
is a run-time error.
```

En ActionScript 3.0, toute tentative d'accéder à une propriété private à l'aide de l'opérateur point (`myExample.privVar`) déclenche une erreur de compilation en mode strict. Autrement, l'erreur est signalée à l'exécution, tout comme lors de l'utilisation de l'opérateur d'accès à la propriété (`myExample["privVar"]`).

Le tableau suivant présente les divers résultats d'une tentative d'accès à une propriété déclarée comme privée qui appartient à une classe scellée (non dynamique) :

	Mode strict	**Mode standard**
opérateur point (`.`)	erreur à la compilation	erreur à l'exécution
opérateur crochet (`[]`)	erreur à l'exécution	erreur à l'exécution

Dans les classes déclarées avec un attribut `dynamic`, les tentatives d'accès à une variable privée ne provoquent pas d'erreur d'exécution. La variable ne sera simplement pas visible, de sorte que Flash Player ou Adobe® AIR™ renvoient la valeur `undefined`. Une erreur à la compilation survient néanmoins si vous utilisez l'opérateur point en mode strict. L'exemple suivant est identique au précédent si ce n'est que la classe PrivateExample est déclarée en tant que classe dynamique :

```
dynamic class PrivateExample
{
    private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);// compile-time error in strict mode
trace(myExample["privVar"]); // output: undefined
```

Les classes dynamiques renvoient le plus souvent la valeur`undefined` plutôt que de générer une erreur lorsque du code, extérieur à une classe, tente d'accéder à une classe déclarée comme propriété privée. Le tableau suivant montre qu'une erreur est générée uniquement lorsque l'opérateur point est utilisé pour accéder à une propriété privée en mode strict :

	Mode strict	Mode standard
opérateur point (`.`)	erreur à la compilation	`undefined`
opérateur crochet (`[]`)	`undefined`	`undefined`

Avec l'attribut `protected`, nouveau en ActionScript 3.0, une propriété n'est visible qu'à partir de sa propre classe ou d'une sous-classe de celle-ci. Autrement dit, une propriété déclarée protected n'est disponible qu'à partir de sa propre classe ou des classes qui lui sont inférieures dans sa hiérarchie d'héritage, que la sous-classe se trouve dans le même package ou dans un autre.

Pour les programmeurs qui connaissent ActionScript 2.0, cette fonctionnalité est semblable à l'attribut `private` d'ActionScript 2.0. L'attribut `protected` d'ActionScript 3.0 est également semblable à l'attribut `protected` de Java, à la différence près que la version de Java permet autorise également l'accès à partir du même package. L'attribut `protected` est utile pour créer une variable ou une méthode nécessaire aux sous-classes, mais qui ne doit pas être visible à partir du code extérieur à la hiérarchie d'héritage.

Avec l'attribut `interne` qui apparaît avec ActionScript 3.0, une propriété n'est visible qu'à partir de son propre package. Il s'agit de l'attribut par défaut pour du code dans un package et il s'applique à toute propriété dépourvue de l'un quelconque des attributs suivants :

- `public`
- `private`
- `protected`
- un espace de noms défini par l'utilisateur

L'attribut `internal` est semblable au contrôle d'accès par défaut de Java quoiqu'en Java il n'existe pas de nom explicite pour ce niveau d'accès ; il ne peut être obtenu que par l'omission de tout autre modificateur d'accès. Avec l'attribut `internal`, qui apparaît avec ActionScript 3.0, il est possible d'indiquer explicitement votre intention de ne rendre une propriété visible qu'à partir de son propre package.

Attribut static

L'attribut `static`, qui peut être utilisé avec des propriétés déclarées à l'aide des mots-clés `var`, `const` ou `function`, vous permet d'associer une propriété à la classe elle-même plutôt qu'à ses occurrences. Le code externe à cette classe doit appeler les propriétés statiques à l'aide du nom de la classe et non pas à partir du nom d'une occurrence.

Les sous-classes n'héritent pas des propriétés statiques ; ces dernières font partie de leur chaîne de portée. En d'autres termes, dans le corps d'une sous-classe, une variable ou une méthode statique peut être utilisée sans faire référence à la classe dans laquelle elle a été définie. Pour plus d'informations, consultez la section « Propriétés statiques non héritées » à la page 117.

Attributs d'espace de noms définis par l'utilisateur

Comme solution de rechange aux attributs de contrôle d'accès prédéfinis, vous pouvez créer un espace de noms personnalisé pour l'utiliser comme attribut. Il ne peut exister qu'un attribut d'espace de noms par définition et vous ne pouvez pas utiliser un tel attribut associé à l'un quelconque des attributs de contrôle d'accès (`public`, `private`, `protected`, `internal`). Pour plus d'informations sur l'utilisation des espaces de noms, consultez la section « Espaces de noms » à la page 44.

Variables

Pour déclarer une variable, utilisez les mots-clés `var` ou `const`. Les variables déclarées avec le mot-clé `var` sont susceptibles de changer de valeur plusieurs fois au cours de l'exécution d'un script. Les variables déclarées à l'aide du mot-clé `const` sont appelées des *constantes* et on ne peut leur attribuer une valeur qu'une seule fois. Une erreur survient lors d'une tentative d'attribution d'une nouvelle valeur à une constante initialisée. Pour plus d'informations, consultez la section « Constantes » à la page 69.

Variables statiques

Les variables statiques sont déclarées à l'aide de l'association du mot-clé `static` et de l'instruction `var` ou `const`. Les variables statiques sont affectées à une classe, plutôt qu'à une occurrence de classe. Elles permettent de stocker et de partager des informations propres à une classe entière d'objets. Par exemple, une variable statique permet d'enregistrer le nombre de fois où une classe a été instanciée ou bien le nombre maximal d'occurrences autorisées pour une classe.

L'exemple ci-dessous crée une variable `totalCount` qui permet d'enregistrer le nombre total d'instanciations d'une classe et une constante `MAX_NUM` dont la valeur est le nombre maximum d'instanciations autorisées. Les variables `totalCount` et `MAX_NUM` sont statiques car elles contiennent des valeurs qui s'appliquent à la classe elle-même, plutôt qu'à une occurrence particulière.

```
class StaticVars
{
    public static var totalCount:int = 0;
    public static const MAX_NUM:uint = 16;
}
```

Un code extérieur à la classe StaticVars, ainsi que toutes ses sous-classes, ne peuvent faire référence aux propriétés `totalCount` et `MAX_NUM` que par le biais de la classe elle-même. Par exemple, le code suivant fonctionne :

```
trace(StaticVars.totalCount); // output: 0
trace(StaticVars.MAX_NUM); // output: 16
```

Comme il est impossible d'accéder à des variables statiques par une occurrence de la classe, le code suivant renvoie des erreurs :

```
var myStaticVars:StaticVars = new StaticVars();
trace(myStaticVars.totalCount); // error
trace(myStaticVars.MAX_NUM); // error
```

Les variables qui sont déclarées avec les deux mots-clés `static` et `const` doivent être initialisées en même temps lors de la déclaration de la constante, tout comme la classe StaticVars le fait pour `MAX_NUM`. Il est impossible d'attribuer une valeur à `MAX_NUM` au sein du constructeur ou d'une méthode d'occurrence. Le code suivant générera une erreur, car ce n'est pas une façon valide d'initialiser une constante statique :

```
// !! Error to initialize static constant this way
class StaticVars2
{
    public static const UNIQUESORT:uint;
    function initializeStatic():void
    {
        UNIQUESORT = 16;
    }
}
```

Variables d'occurrence

Les variables d'occurrence contiennent des propriétés déclarées à l'aide des mots-clés `var` et `const`, mais sans le mot-clé `static`. Les variables d'occurrence, qui sont associées à des occurrences de classe plutôt qu'à la classe elle-même, sont utiles pour conserver des valeurs spécifiques à une occurrence. Par exemple, la classe Array dispose d'une propriété d'occurrence appelée `length` qui conserve le nombre d'éléments du tableau appartenant à une occurrence particulière de la classe Array.

Qu'elles soient déclarées avec le mot-clé `var` ou `const`, les variables d'occurrence ne peuvent pas être redéfinies dans une sous-classe. Il est toutefois possible d'obtenir un effet similaire à la redéfinition de variables, en redéfinissant des méthodes de lecture et de définition. Pour plus d'informations, consultez la section « Méthodes accesseur get et set » à la page 104.

Méthodes

Les méthodes sont des fonctions associées à la définition d'une classe. Dès la création d'une occurrence de la classe, une méthode est liée à cette occurrence. Contrairement à une fonction déclarée hors d'une classe, une méthode ne peut pas être utilisée séparément de l'occurrence à laquelle elle est associée.

Les méthodes sont définies à l'aide du mot-clé `function`. Comme toute propriété de classe, vous pouvez appliquer n'importe quel attribut de propriété de classe aux méthodes, qu'elles soient privées, protégées, publiques, internes ou statiques, ainsi qu'à un espace de noms personnalisé. Vous pouvez utiliser une instruction de fonction telle que :

```
public function sampleFunction():String {}
```

Vous pouvez aussi utiliser une variable à laquelle vous attribuez une expression de fonction, comme ci-dessous :

```
public var sampleFunction:Function = function () {}
```

Dans la plupart des cas, il est préférable d'utiliser utiliser une instruction function qu'une expression fonction pour les raisons suivantes :

- Les instructions function sont plus concises et plus faciles à lire.
- Elles vous permettent d'utiliser les mots-clés `override` et `final` . Pour plus d'informations, consultez la section « Redéfinition des méthodes » à la page 115.
- Les instructions function créent une liaison plus robuste entre l'identificateur (le nom de la fonction) et le code dans le corps de la méthode. Comme la valeur d'une variable peut être modifiée par une instruction assignment, le lien entre une variable et son expression fonction peut être rompu à tout moment. Bien qu'il soit possible de résoudre ce problème en déclarant la variable avec `const` au lieu de `var`, cette technique n'est pas recommandée car elle rend le code difficilement lisible et empêche d'utiliser les mots-clés `override` et `final`.

Il existe toutefois un cas dans lequel une expression de fonction doit être utilisée : si vous choisissez d'affecter une fonction à l'objet prototype. Pour plus d'informations, consultez la section « Objet prototype » à la page 123.

Méthodes constructeur

Les méthodes de constructeur, parfois appelées simplement *constructeurs*, sont des fonctions qui portent le nom de la classe dans laquelle elles sont définies. Tout code qui figure dans une méthode constructeur est exécuté toutes les fois qu'une occurrence de la classe est créée à l'aide du mot-clé `new`. Par exemple, le code suivant définit une classe simple appelée Example qui contient une propriété unique appelée `status`. La valeur initiale de la variable `status` est fixée dans la fonction constructeur.

```
class Example
{
    public var status:String;
    public function Example()
    {
        status = "initialized";
    }
}

var myExample:Example = new Example();
trace(myExample.status); // output: initialized
```

Les méthodes constructeur ne peuvent être que publiques, mais l'utilisation de l'attribut `public` est facultative. Il est impossible d'utiliser l'un des autres spécificateurs de contrôle d'accès, y compris `private`, `protected` ou `internal` avec un constructeur. De même qu'il est impossible d'utiliser non plus, avec un constructeur, un espace de noms défini par l'utilisateur.

Un constructeur peut appeler explicitement le constructeur de sa superclasse directe, à l'aide de l'instruction `super()`. Si le constructeur de la superclasse n'est pas explicitement appelé, le compilateur insère automatiquement un appel devant la première instruction dans le corps du constructeur. Vous pouvez aussi appeler des méthodes de la superclasse à l'aide du préfixe `super` en référence à la superclasse. Si vous décidez d'utiliser à la fois `super()` et `super` dans le corps du même constructeur, veillez à appeler `super()` en premier. Sinon, la référence `super` n'aura pas le comportement prévu. Le constructeur `super()` devrait également être appelé avant toute instruction `throw` ou `return`.

L'exemple suivant décrit ce qui se passe si vous tentez d'utiliser la référence `super` avant d'appeler le constructeur `super()`. Une nouvelle classe, ExampleEx, étend la classe Example. Le constructeur ExampleEx tente d'accéder à la variable d'état définie dans sa super classe, mais avant un appel à `super()`. L'instruction `trace()` du constructeur ExampleEx produit la valeur `null` car la variable `status` n'est pas disponible tant que le constructeur `super()` n'a pas été exécuté.

```
class ExampleEx extends Example
{
    public function ExampleEx()
    {
        trace(super.status);
        super();
    }
}

var mySample:ExampleEx = new ExampleEx(); // output: null
```

Bien que l'utilisation de l'instruction `return` au sein d'un constructeur soit autorisée, il n'est pas permis de renvoyer une valeur. En d'autres termes, aucune expression ou valeur ne peut être associée à l'instruction `return`. Par conséquent, les méthodes constructeur ne sont pas autorisées à renvoyer des valeurs, ce qui signifie qu'aucun type de valeur renvoyée ne peut être spécifié.

Si vous ne définissez pas de méthode constructeur dans votre classe, le compilateur créera automatiquement un constructeur vide. Si votre classe étend une autre classe, le compilateur insère un appel `super()` dans le constructeur qu'il génère.

Méthodes statiques

Les méthodes statiques, également appelées parfois *méthodes de classe*, sont déclarées avec le mot-clé `static`. Les méthodes statiques sont affectées à une classe plutôt qu'à une occurrence de classe. Elles permettent d'encapsuler des fonctionnalités qui ont une portée plus étendue que l'état d'une occurrence individuelle. Comme les méthodes statiques sont associées à l'intégralité d'une classe, on peut accéder à des méthodes statiques uniquement par une classe et pas du tout par une occurrence de classe.

Les méthodes statiques permettent d'encapsuler des fonctionnalités qui ne se bornent pas à la modification d'état des occurrences de classe. Autrement dit, une méthode devrait être statique si elle offre des fonctionnalités qui n'affectent pas directement la valeur d'une occurrence de classe. Par exemple, la classe Date possède une méthode statique appelée `parse()` qui reçoit une chaîne et la convertit en nombre. La méthode est statique parce qu'elle n'affecte pas une occurrence individuelle de sa classe. La méthode `parse()`, reçoit une chaîne représentant une valeur de date, l'analyse et renvoie un nombre dans un format compatible avec la représentation interne d'un objet Date. Cette méthode n'est pas une méthode d'occurrence, puisqu'il n'y aurait aucun intérêt à l'appliquer à une occurrence de la classe Date.

Comparons la méthode statique `parse()` à l'une des méthodes d'occurrence de la classe Date, comme `getMonth()`. La méthode `getMonth()` est une méthode d'occurrence parce qu'elle agit directement sur la valeur d'une occurrence en récupérant un composant spécifique, le mois, d'une occurrence de Date.

Comme les méthodes statiques ne sont pas liées à des occurrences individuelles, vous ne pouvez pas utiliser les mots-clés `this` ou `super` dans le corps d'une méthode statique. Les deux références `this` et `super` n'ont de sens que dans le contexte d'une méthode d'occurrence.

Contrairement à d'autres langages de programmation basés sur des classes, les méthodes statiques en ActionScript 3.0 ne sont pas héritées. Pour plus d'informations, consultez la section « Propriétés statiques non héritées » à la page 117.

Méthodes d'occurrence

Les méthodes d'occurrence sont déclarées sans le mot-clé `static`. Les méthodes d'occurrence, qui sont affectées aux occurrences d'une classe et non pas à la classe elle-même, permettent d'implémenter des fonctionnalités qui affectent des occurrences individuelles d'une classe. Par exemple, la classe Array contient une méthode d'occurrence appelée `sort()` qui opère directement sur des occurrences Array.

Dans le corps d'une méthode d'occurrence, les variables statiques et d'occurrence sont de même portée ; ce qui signifie que les variables définies dans la même classe peuvent être référencées à l'aide d'un identificateur simple. Par exemple, la classe suivante, CustomArray, étend la classe Array. La classe CustomArray définit une variable statique appelée `arrayCountTotal` destinée à contenir le nombre total d'occurrences de la classe, une variable d'occurrence appelée `arrayNumber` qui enregistre l'ordre dans lequel les occurrences ont été créées et une méthode d'occurrence appelée `getPosition()` qui renvoie les valeurs de ces variables.

```
public class CustomArray extends Array
{
    public static var arrayCountTotal:int = 0;
    public var arrayNumber:int;

    public function CustomArray()
    {
        arrayNumber = ++arrayCountTotal;
    }

    public function getArrayPosition():String
    {
         return ("Array " + arrayNumber + " of " + arrayCountTotal);
    }
}
```

Pour faire référence à la variable statique `arrayCountTotal`, du code externe à cette classe doit passer par l'objet class `CustomArray.arrayCountTotal` ; mais le code qui réside dans le corps de la méthode `getPosition()` peut faire référence directement à la variable statique `arrayCountTotal`. C'est également le cas pour les variables statiques dans les superclasses. Bien que les propriétés statiques ne soient pas héritées en ActionScript 3.0, les propriétés statiques des superclasses sont dans la portée. Par exemple, la classe Array possède quelques variables statiques, dont l'une est une constante appelée `DESCENDING`. Le code qui réside dans une sous-classe d'Array peut faire référence à la constante statique `DESCENDING` à l'aide d'un identificateur simple :

```
public class CustomArray extends Array
{
    public function testStatic():void
    {
        trace(DESCENDING); // output: 2
    }
}
```

La valeur de la référence `this` dans le corps d'une méthode d'occurrence est une référence à l'occurrence à laquelle la méthode est affectée. Le code suivant montre que la référence `this` pointe sur l'occurrence qui contient la méthode :

```
class ThisTest
{
    function thisValue():ThisTest
    {
        return this;
    }
}

var myTest:ThisTest = new ThisTest();
trace(myTest.thisValue() == myTest); // output: true
```

Il est possible de contrôler l'héritage des méthodes d'occurrence à l'aide des mots-clés `override` et `final`. Vous pouvez utiliser l'attribut `override` pour redéfinir une méthode héritée et l'attribut `final` pour empêcher les sous-classes de redéfinir une méthode. Pour plus d'informations, consultez la section « Redéfinition des méthodes » à la page 115.

Méthodes accesseur get et set

Les fonctions d'accesseur de lecture et de définition, appelées aussi *getters* et *setters*, vous permettent de suivre les principes de programmation sur le masquage et l'encapsulation des informations tout en offrant une interface de programmation conviviale pour les classes que vous créez. Les fonctions de lecture et de définition (get et set) permettent de garder privées les propriétés d'une classe. Par contre, elles permettent à des utilisateurs de votre classe d'accéder à ces propriétés comme s'ils accédaient à une variable de classe au lieu d'appeler une méthode de classe.

L'avantage de cette approche est qu'elle permet d'éviter les fonctions d'accesseur traditionnelles aux noms compliqués, comme `getPropertyName()` et `setPropertyName()`. Leur autre avantage est qu'elles évitent d'avoir deux fonctions exposées publiquement pour chaque propriété accessible en lecture et en écriture.

Dans l'exemple suivant, la classe appelée GetSet, possède des fonctions accesseurs de lecture et de définition appelées `publicAccess()` qui permettent d'accéder à la variable privée appelée `privateProperty` :

```
class GetSet
{
    private var privateProperty:String;

    public function get publicAccess():String
    {
        return privateProperty;
    }

    public function set publicAccess(setValue:String):void
    {
        privateProperty = setValue;
    }
}
```

Si vous tentez d'accéder directement à la propriété `privateProperty`, une erreur se produira :

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.privateProperty); // error occurs
```

Par contre, si vous utilisez la classe GetSet, vous ferez appel à quelque chose qui paraît être une propriété appelée `publicAccess` ; mais il s'agit là d'une paire de fonctions accesseurs de lecture et de définition qui opèrent sur la propriété privée appelée `privateProperty`. L'exemple suivant instancie la classe GetSet, puis définit la valeur de la propriété `privateProperty` à l'aide de l'accesseur public appelé `publicAccess` :

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.publicAccess); // output: null
myGetSet.publicAccess = "hello";
trace(myGetSet.publicAccess); // output: hello
```

Les fonctions getter et setter permettent également de redéfinir des propriétés héritées d'une superclasse, ce qui n'est pas possible avec des variables régulières membres de classes. Les variables des membres de la classe qui sont déclarées à l'aide du mot-clé `var` ne peuvent pas être redéfinies dans une sous-classe. Toutefois, cette restriction ne concerne pas les propriétés créées à l'aide des fonctions getter et setter. Vous pouvez utiliser l'attribut `override` sur des fonctions getter et setter héritées d'une superclasse.

Méthodes liées

Une méthode liée, parfois appelée *fermeture de méthode*, est tout simplement une méthode extraite de son occurrence. On peut citer comme exemples les méthodes passées en arguments à une fonction ou renvoyées comme valeurs par une fonction. La méthode liée, qui est une nouveauté d'ActionScript 3.0, est semblable à une fermeture de fonction dans la mesure où elle conserve son environnement lexical, même après avoir été extraite de son occurrence. Toutefois, la différence entre une méthode liée et une fermeture de fonction réside dans le fait que la référence `this` d'une méthode liée reste liée à l'occurrence qui implémente cette méthode. Autrement dit, la référence `this` d'une méthode liée pointe toujours sur l'objet original qui a implémenté la méthode. Pour les fermetures de fonction, la référence `this` est générique, ce qui signifie qu'elle pointe sur l'objet auquel est associée la fonction lorsqu'elle est appelée.

Il est important de comprendre les méthodes liées pour utiliser le mot-clé `this` à bon escient. N'oubliez pas que `this` représente une référence à l'objet parent d'une méthode. La plupart des programmeurs en ActionScript s'attendent à ce que le mot-clé `this` fasse toujours référence à l'objet ou à la classe qui contient la définition d'une méthode. Ce n'est pas toujours le cas sans méthode liée. Par exemple, dans les versions précédentes d'ActionScript, la référence `this` ne pointait pas toujours sur l'occurrence qui implémentait la méthode. En ActionScript 2.0, lorsque les méthodes sont

extraites d'une occurrence, non seulement la référence `this` n'est pas liée à l'occurrence originale, mais les variables et les méthodes de la classe de cette occurrence ne sont pas disponibles. Toutefois, ce problème n'existe plus avec ActionScript 3.0, car les méthodes liées sont automatiquement créées lorsque la méthode est passée en paramètre. Avec les méthodes liées, le mot-clé `this` fait toujours référence à l'objet ou à la classe dans laquelle la méthode est définie.

Le code suivant définit une classe appelée ThisTest, qui contient une méthode appelée `foo()` définissant la méthode liée et une méthode appelée `bar()` qui renvoie cette méthode liée. Le code extérieur à la classe crée une occurrence de la classe ThisTest, appelle la méthode `bar()` et enregistre la valeur à renvoyer dans la variable `myFunc`.

```
class ThisTest
{
    private var num:Number = 3;
    function foo():void // bound method defined
    {
        trace("foo's this: " + this);
        trace("num: " + num);
    }
    function bar():Function
    {
        return foo; // bound method returned
    }
}

var myTest:ThisTest = new ThisTest();
var myFunc:Function = myTest.bar();
trace(this); // output: [object global]
myFunc();
/* output:
foo's this: [object ThisTest]
output: num: 3 */
```

Les deux dernières lignes de code montrent que la référence `this` dans la méthode liée `foo()` pointe encore sur une occurrence de la classe ThisTest, bien que la référence `this` de la ligne précédente pointe sur l'objet global. De plus, la méthode liée, stockée dans la variable `myFunc`, peut encore accéder aux variables membres de la classe ThisTest. Si ce code est exécuté en ActionScript 2.0, les références `this` seront identiques et la variable `num` sera `undefined`.

Les gestionnaires d'événement constituent un domaine dans lequel l'ajout des méthodes liées est le plus notable, car la méthode `addEventListener()` nécessite de transmettre une fonction ou une méthode en argument. Pour plus d'informations, consultez la section Fonction d'écouteur définie comme méthode de classe dans « Les écouteurs d'événement » à la page 264.

Enumérations et classes

Les *énumérations* sont des types de données que vous pouvez créer pour encapsuler un petit ensemble de valeur. Contrairement à C++ avec le mot-clé `enum` et à Java avec l'interface d'énumération, ActionScript 3.0 ne dispose pas d'un mécanisme d'énumération spécifique. Il est toutefois possible de créer des énumérations à l'aide de classes et de constantes statiques. par exemple, la classe PrintJob dans ActionScript 3.0 utilise une énumération appelée PrintJobOrientation pour stocker le jeu de valeurs comprenant `"landscape"` et `"portrait"`, comme l'indique le code ci-dessous :

```
public final class PrintJobOrientation
{
    public static const LANDSCAPE:String = "landscape";
    public static const PORTRAIT:String = "portrait";
}
```

Par convention, une classe d'énumération est déclarée avec l'attribut `final`, car il n'est pas nécessaire d'étendre cette classe. Cette classe étant composée uniquement de membres statiques, il n'est pas possible d'en créer des occurrences. En effet, on accède aux valeurs de l'énumération directement par l'objet classe, comme le montre l'extrait de code suivant :

```
var pj:PrintJob = new PrintJob();
if(pj.start())
{
    if (pj.orientation == PrintJobOrientation.PORTRAIT)
    {
        ...
    }
    ...
}
```

Toutes les classes d'énumération d'ActionScript 3.0 contiennent uniquement des variables de type String, int ou uint. Le fait que les fautes de frappe sont plus faciles à détecter avec les énumérations présente un grand avantage par rapport à des chaînes littérales ou des nombres. Si vous faites une erreur dans le nom d'une énumération, le compilateur ActionScript génère une erreur. Si vous utilisez des valeurs littérales, le compilateur acceptera un nom mal épelé ou un chiffre erroné. Dans l'exemple ci-dessus, le compilateur génère une erreur si le nom de la constante d'énumération est incorrect, comme dans l'extrait suivant :

```
    if (pj.orientation == PrintJobOrientation.PORTRAI) // compiler error
```

Toutefois, le compilateur ne génère pas d'erreur si vous faites une faute de frappe dans le nom d'une chaîne littérale :

```
    if (pj.orientation == "portrai") // no compiler error
```

Une autre technique de création d'énumérations consiste à créer une classe séparée avec des propriétés statiques pour l'énumération. Toutefois, cette technique est différente dans la mesure où chacune des propriétés statiques contient une occurrence de la classe plutôt qu'une valeur chaîne ou un entier. Par exemple, le code suivant crée une classe d'énumération pour les jours de la semaine :

```
public final class Day
{
    public static const MONDAY:Day = new Day();
    public static const TUESDAY:Day = new Day();
    public static const WEDNESDAY:Day = new Day();
    public static const THURSDAY:Day = new Day();
    public static const FRIDAY:Day = new Day();
    public static const SATURDAY:Day = new Day();
    public static const SUNDAY:Day = new Day();
}
```

Cette technique n'est pas utilisée par ActionScript 3.0, mais de nombreux développeurs y ont recours car elle permet d'obtenir un meilleur type de vérification. Par exemple, une méthode qui renvoie une valeur d'énumération peut restreindre la valeur renvoyée au type de données de l'énumération. Le code suivant illustre non seulement une fonction qui renvoie un jour de la semaine, mais aussi un appel de fonction qui utilise le type énumération comme annotation de type :

```
function getDay():Day
{
    var date:Date = new Date();
    var retDay:Day;
    switch (date.day)
    {
        case 0:
            retDay = Day.MONDAY;
            break;
        case 1:
            retDay = Day.TUESDAY;
            break;
        case 2:
            retDay = Day.WEDNESDAY;
            break;
        case 3:
            retDay = Day.THURSDAY;
            break;
        case 4:
            retDay = Day.FRIDAY;
            break;
        case 5:
            retDay = Day.SATURDAY;
            break;
        case 6:
            retDay = Day.SUNDAY;
            break;
    }
    return retDay;
}

var dayOfWeek:Day = getDay();
```

Il est également possible d'améliorer la classe Day afin qu'elle associe un entier à chaque jour de la semaine. Elle comporte une méthode `toString()` renvoyant une représentation du jour sous forme de chaîne. Il est conseillé de créer cette amélioration comme exercice.

Classes des éléments incorporés

ActionScript 3.0 utilise des classes spéciales, appelées *classes des éléments incorporés*, pour représenter les éléments incorporés. Un *élément incorporé* est un élément (son, image ou police) qui est incorporé au fichier SWF lors de la compilation. Contrairement au chargement dynamique, l'incorporation des éléments les rend disponibles immédiatement lors de l'exécution ; mais cette méthode augmente la taille des fichiers SWF.

Utilisation de classes d'éléments incorporés avec Flash

Pour incorporer un élément, placez-le d'abord dans la bibliothèque d'un fichier FLA. Utilisez ensuite la propriété de liaison de l'élément pour fournir un nom à la classe de l'élément incorporé. S'il n'existe pas de classe de ce nom dans le chemin de classe indiqué, une classe est automatiquement créée. Vous pouvez alors créer une occurrence de la classe d'éléments incorporés et utiliser les propriétés et méthodes définies ou héritées par cette classe. Par exemple, le code suivant permet de lire un son intégré et lié à une classe d'éléments incorporés appelée PianoMusic :

```
var piano:PianoMusic = new PianoMusic();
var sndChannel:SoundChannel = piano.play();
```

Interfaces

Une interface est une collection de déclarations de méthodes qui autorise les communications entre des objets différents. Par exemple, ActionScript 3.0 définit l'interface IEventDispatcher qui contient les déclarations des méthodes qu'une classe peut utiliser pour gérer les objets événements. L'interface IEventDispatcher établit une technique standard permettant aux objets de s'échanger les événements. Le code suivant représente la définition de l'interface IEventDispatcher:

```
public interface IEventDispatcher
{
    function addEventListener(type:String, listener:Function,
            useCapture:Boolean=false, priority:int=0,
            useWeakReference:Boolean = false):void;
    function removeEventListener(type:String, listener:Function,
            useCapture:Boolean=false):void;
    function dispatchEvent(event:Event):Boolean;
    function hasEventListener(type:String):Boolean;
    function willTrigger(type:String):Boolean;
}
```

Les interfaces sont basées sur la distinction entre l'interface d'une méthode et l'implémentation de celle-ci. L'interface d'une méthode comprend toutes les informations nécessaires pour appeler cette méthode : le nom de la méthode, l'ensemble des paramètres qu'elle reçoit et le type de données qu'elle renvoie. L'implémentation d'une méthode comprend non seulement les informations de l'interface, mais aussi les instructions exécutables qui caractérisent le comportement de la méthode. La définition d'une interface ne contient que les interfaces de la méthode et toute classe qui implémente l'interface doit donc définir les implémentations de la méthode.

Dans ActionScript 3.0, la classe EventDispatcher implémente l'interface IEventDispatcher en définissant toutes les méthodes de cette interface et en ajoutant le corps de chacune de ces méthodes. Le code suivant est extrait de la définition de la classe EventDispatcher :

```
public class EventDispatcher implements IEventDispatcher
{
    function dispatchEvent(event:Event):Boolean
    {
        /* implementation statements */
    }

    ...
}
```

L'interface IEventDispatcher fait office de protocole utilisé par les occurrences d'EventDispatcher pour traiter les objets événements et les transmettre aux autres objets qui ont également implémenté cette interface.

Il est aussi possible de décrire une interface en disant qu'elle définit un type de données, au même titre qu'une classe. En conséquence, une interface peut être utilisée comme annotation de type, tout comme une classe. En tant que type de données, une interface peut également être utilisée avec des opérateurs, par exemple les opérateurs `is` et `as`, qui nécessitent un type de données. Toutefois, à l'inverse d'une classe, il n'est pas possible d'instancier une interface. C'est en raison de cette distinction que de nombreux programmeurs voient les interfaces comme des types de données abstraites et les classes comme des types de données concrètes.

Définition d'une interface

La structure de la définition d'une interface est similaire à celle de la définition d'une classe, à ceci près qu'une interface ne peut pas contenir les corps des méthodes. Les interfaces ne peuvent pas comporter des variables ou des constantes, mais elles peuvent contenir des méthodes de lecture et de définition. Pour définir une interface, on utilise le mot-clé `interface`. Par exemple, l'interface suivante, IExternalizable, fait partie du package flash.utils d'ActionScript 3.0. L'interface IExternalizable définit un protocole pour sérialiser un objet, ce qui correspond à la conversion d'un objet en un format qui convienne au stockage sur un périphérique ou au transport sur un réseau.

```
public interface IExternalizable
{
    function writeExternal(output:IDataOutput):void;
    function readExternal(input:IDataInput):void;
}
```

Vous pouvez remarquer que l'interface IExternalizable est déclarée avec le modificateur d'accès `public`. Les définitions d'interfaces ne peuvent être modifiées qu'à l'aide des spécificateurs de contrôle d'accès `public` et `internal`. Dans une définition d'interface, les déclarations de méthodes ne peuvent pas comporter de spécificateur de contrôle d'accès.

ActionScript 3.0 respecte la convention de nom selon laquelle les noms des interfaces débutent par un `I` majuscule, mais vous pouvez utiliser tout identificateur autorisé comme nom d'interface. Les définitions d'interfaces sont souvent placées au niveau supérieur d'un package. Les définitions d'interfaces ne peuvent pas être placées dans une définition de classe ou dans une autre définition d'interface.

Une interface peut étendre une ou plusieurs autres interfaces. Par exemple, l'interface IExample étend l'interface IExternalizable :

```
public interface IExample extends IExternalizable
{
    function extra():void;
}
```

Toute classe qui implémente l'interface IExample doit comporter non seulement les implémentations de la méthode `extra()`, mais aussi celles des méthodes `writeExternal()` et readExternal() héritées de l'interface IExternalizable.

Implémentation d'une interface dans une classe

La classe est le seul élément du langage ActionScript 3.0 qui puisse implémenter une interface. Pour implémenter une ou plusieurs interfaces, on utilise le mot-clé `implements`dans une déclaration de classe. L'exemple suivant définit deux interfaces, IAlpha et IBeta, ainsi qu'une classe, Alpha, qui les implémente toutes deux :

```
interface IAlpha
{
    function foo(str:String):String;
}

interface IBeta
{
    function bar():void;
}

class Alpha implements IAlpha, IBeta
{
    public function foo(param:String):String {}
    public function bar():void {}
}
```

Dans une classe qui implémente une interface, les méthodes implémentées doivent :

- Utiliser l'identificateur de contrôle d'accès `public`.
- Utiliser le même nom que la méthode de l'interface.
- Avoir le même nombre de paramètres, chacun d'eux étant du type de données correspondant à celui du paramètre équivalent dans la méthode de l'interface.
- Utiliser le même type de retour.

```
public function foo(param:String):String {}
```

Vous disposez toutefois d'une certaines souplesse pour le nom des paramètres des méthodes que vous implémentez. Bien que le nombre et le type de données des paramètres de la méthode implémentée doivent correspondre à ceux de la méthode de l'interface, il n'est pas obligatoire que les noms des paramètres soient identiques. Par exemple, dans l'exemple ci-dessus, le paramètre de la méthode `Alpha.foo()` est appelé `param`.

Par contre, le paramètre correspondant est appelé `str` dans la méthode de l'interface `IAlpha.foo()` :

```
function foo(str:String):String;
```

Les valeurs par défaut des paramètres offrent également une certaine souplesse. La définition d'une interface peut comporter des déclarations de fonctions avec des valeurs par défaut pour les paramètres. Une méthode qui implémente l'une de ces déclarations de fonction doit disposer d'une valeur par défaut pour le ou les paramètres. Cette valeur doit être du même type de données que celle qui est spécifiée dans la définition de l'interface, mais ce n'est pas forcément le cas pour la valeur réelle. Par exemple, le code ci-dessous définit une interface contenant une méthode dont le paramètre a la valeur 3 par défaut :

```
interface IGamma
{
    function doSomething(param:int = 3):void;
}
```

La définition de classe suivante implémente l'interface Igamma, mais utilise une autre valeur par défaut pour le paramètre :

```
class Gamma implements IGamma
{
    public function doSomething(param:int = 4):void {}
}
```

Cette souplesse est due au fait que les règles d'implémentation d'une interface sont spécifiquement conçues afin d'assurer une compatibilité des types de données ; il n'est pas nécessaire, pour ce faire, d'exiger des noms et des valeurs par défaut identiques pour les paramètres.

Héritage

L'héritage est une forme de réutilisation du code qui permet aux programmeurs de développer de nouvelles classes à partir de classes existantes. Les classes existantes sont alors fréquemment appelées *classes de base* ou *superclasses*, alors que les nouvelles classes sont généralement appelées *sous-classes*. L'un des principaux avantages de l'héritage est qu'il permet de réutiliser le code d'une classe de base sans modifier le code existant. De plus, l'héritage ne nécessite pas de modifier les modes d'interaction des autres classes avec la classe de base. Plutôt que de modifier une classe existante, qui est peut-être soigneusement testée et déjà utilisée, l'héritage permet de traiter cette classe comme un module intégré qui peut être étendu à l'aide de propriétés et de méthodes supplémentaires. C'est pourquoi on utilise le mot-clé `extends` pour indiquer qu'une classe hérite d'une autre classe.

L'héritage permet également de tirer parti du *polymorphisme* du code. Le polymorphisme est la possibilité d'utiliser un nom de méthode unique pour une méthode qui se comporte différemment en fonction des types de données qu'elle reçoit. Par exemple, supposons une classe de base appelée Shape, avec deux sous-classes appelées Circle et Square. La classe Shape définit une méthode appelée `area()` qui renvoie la surface de Shape. Si vous avez implémenté le polymorphisme, vous pouvez appeler la méthode `area()` pour les objets de type Circle ou Square et lui faire exécuter le calcul correct. L'héritage autorise le polymorphisme en permettant aux sous-classes d'hériter et de définir, ou *override*, les méthodes de la classe de base. Dans l'exemple suivant, la méthode `area()` est redéfinie par les classes Circle et Square :

```
class Shape
{
    public function area():Number
    {
        return NaN;
    }
}

class Circle extends Shape
{
    private var radius:Number = 1;
    override public function area():Number
    {
        return (Math.PI * (radius * radius));
    }
}

class Square extends Shape
{
    private var side:Number = 1;
    override public function area():Number
    {
        return (side * side);
    }
}

var cir:Circle = new Circle();
trace(cir.area()); // output: 3.141592653589793
var sq:Square = new Square();
trace(sq.area()); // output: 1
```

Dans la mesure où chaque classe définit un type de données, l'utilisation de l'héritage crée un rapport spécial entre une classe de base et une classe qui l'étend. Une sous-classe possède obligatoirement toutes les propriétés de sa classe de base, ce qui signifie qu'il est toujours possible de substituer une occurrence d'une sous-classe à une occurrence de la classe de base. Par exemple, si une méthode définit un paramètre du type Shape, il est parfaitement valable de lui transmettre un argument du type Circle, car Circle étend Shape, comme on peut le voir ci-dessous :

```
function draw(shapeToDraw:Shape) {}

var myCircle:Circle = new Circle();
draw(myCircle);
```

Propriétés et héritage des occurrences

Qu'elle soit définie à l'aide du mot-clé `function`, `var` ou `const`, une propriété d'occurrence est héritée par toutes les sous-classes tant que cette propriété n'est pas déclarée avec l'attribut `private` dans la classe de base. Par exemple, la classe Event d'ActionScript 3.0 possède des sous-classes nombreuses qui héritent de propriétés communes à tous les objets événements.

Pour certains types d'événements, la classe Event contient toutes les propriétés nécessaires pour définir l'événement. Ces types d'événements ne nécessitent pas de propriétés d'occurrence au-delà de celles qui sont définies dans la classe Event. L'événement `complete`, qui est déclenché lorsque des données ont été chargées avec succès, et l'événement `connect`, qui se produit lorsqu'une connexion réseau a été établie, sont des exemples de ce type d'événement.

L'exemple suivant est extrait de la classe Event. Il montre certaines propriétés et méthodes dont les sous-classes héritent. Comme elles sont héritées, ces propriétés sont accessibles par une occurrence de n'importe quelle sous-classe.

```
public class Event
{
    public function get type():String;
    public function get bubbles():Boolean;
    ...

    public function stopPropagation():void {}
    public function stopImmediatePropagation():void {}
    public function preventDefault():void {}
    public function isDefaultPrevented():Boolean {}
    ...
}
```

D'autres types d'événements nécessitent des propriétés uniques qui ne sont pas disponibles dans la classe Event. Ces événements sont définis à l'aide de sous-classes de la classe Event, ce qui permet d'ajouter de nouvelles propriétés à celles de cette classe Event. La classe MouseEvent est un exemple de sous-classe de ce type. Elle ajoute des propriétés uniques aux événements associés à un mouvement ou à un clic de souris, tels que les événements `mouseMove` et `click`. L'exemple suivant est extrait de la classe MouseEvent. Il montre la définition des propriétés inhérentes à la sous-classe parce qu'absentes de la classe de base.

```
public class MouseEvent extends Event
{
    public static const CLICK:String= "click";
    public static const MOUSE_MOVE:String = "mouseMove";
    ...

    public function get stageX():Number {}
    public function get stageY():Number {}
    ...
}
```

Spécificateurs de contrôle d'accès et héritage

Si une propriété est déclarée avec le mot-clé `public`, elle est visible de n'importe quel point du code. Cela signifie que le mot-clé `public` n'introduit aucune restriction sur l'héritage des propriétés, contrairement aux mots-clés `private`, `protected` et `internal`.

Si une propriété est déclarée avec le mot-clé `private`, elle n'est visible qu'à partir de la classe qui la définit. Autrement dit les sous-classes n'en héritent pas. Ce comportement est différent de celui des versions antérieures d'ActionScript où le mot-clé `private` se comportait plutôt comme le mot-clé `protected` d'ActionScript 3.0.

Le mot-clé `protected` indique qu'une propriété est visible non seulement à partir de la classe qui la définit, mais aussi à partir de toutes les sous-classes de celle-ci. Contrairement au mot-clé `protected` en Java, en ActionScript 3.0 le mot-clé`protected` ne rend pas une propriété visible à partir de toutes les autres classes du même package. En ActionScript 3.0, seules les sous-classes peuvent accéder à une propriété déclarée avec le mot-clé `protected`. De plus, une propriété protégée est visible à partir d'une sous-classe même si celle-ci ne se trouve pas dans le même package que sa classe de base.

Pour limiter la visibilité d'une propriété au package dans lequel elle est définie, vous pouvez soit utiliser le mot-clé `internal`, soit n'utiliser aucun spécificateur de contrôle d'accès. Le spécificateur de contrôle d'accès `internal` est appliqué par défaut si vous n'en indiquez aucun. Seule une sous-classe résidant dans le même package pourra hériter d'une propriété marquée comme internal. Seule une sous-classe résidant dans le même package pourra hériter dune propriété désignée comme `internal`.

L'exemple suivant montre comment chaque spécificateur de contrôle d'accès affecte l'héritage dans et au-delà des package. Le code ci-dessous définit une classe principale d'application appelée AccessControl et deux autres classes, Base et Extender. La classe Base se trouve dans un package appelé foo et la classe Extender, qui est une sous-classe de la classe Base, dans un package appelé bar. La classe AccessControl n'importe que la classe Extender et crée une occurrence de celle-ci qui tente d'accéder à une variable appelée `str`, définie dans la classe Base. La variable `str` est déclarée comme `public`, si bien que le code est compilé et exécuté comme l'illustre l'exemple ci-dessous :

```
// Base.as in a folder named foo
package foo
{
    public class Base
    {
        public var str:String = "hello"; // change public on this line
    }
}

// Extender.as in a folder named bar
package bar
{
    import foo.Base;
    public class Extender extends Base
    {
        public function getString():String {
            return str;
        }
    }
}

// main application class in file named AccessControl.as
package
{
    import flash.display.MovieClip;
    import bar.Extender;
    public class AccessControl extends MovieClip
    {
        public function AccessControl()
        {
            var myExt:Extender = new Extender();
            trace(myExt.str);// error if str is not public
            trace(myExt.getString()); // error if str is private or internal
        }
    }
}
```

Pour voir l'effet des autres spécificateurs de contrôle d'accès lors de la compilation et de l'exécution de cet exemple, changez le spécificateur de contrôle de la variable `str` à `private`, `protected` ou `internal` après avoir supprimé ou mis en commentaire la ligne suivante dans la classe `AccessControl` :

```
trace(myExt.str);// error if str is not public
```

Redéfinition des variables impossible

Les propriétés déclarées à l'aide des mots-clés `var` ou `const` sont héritées mais ne peuvent pas être redéfinies. Redéfinir, ou forcer, une propriété au sens de « override ».signifie redéfinir cette propriété dans une sous-classe. Les méthodes (c'est à dire les propriétés déclarées avec le mot-clé `function`) constituent le seul type de propriété qu'il est possible de redéfinir. Bien qu'il soit impossible de redéfinir une variable d'occurrence, vous pouvez obtenir le même résultat par la création des méthodes de lecture et de définition pour cette variable d'occurrence et en forçant ces méthodes. Pour plus d'informations, consultez la section « Redéfinition des méthodes » à la page 115.

Redéfinition des méthodes

Redéfinir une méthode signifie redéfinir le comportement d'une méthode héritée. Les méthodes statiques ne sont pas héritées et ne peuvent donc pas être redéfinies. Toutefois, les sous-classes héritent des méthodes d'occurrence et il est donc possible de redéfinir celles-ci sous réserve de deux conditions :

- La méthode d'occurrence ne doit pas être déclarée avec le mot-clé `final` dans la classe de base. Lorsqu'il est utilisé avec une méthode d'occurrence, le mot-clé `final` indique que le programmeur veut empêcher les sous-classes de redéfinir cette méthode.
- La méthode d'occurrence ne doit pas être déclarée avec le contrôle d'accès `private` dans la classe de base. Si une méthode est désignée comme `private` dans la classe de base, il n'est pas nécessaire d'utiliser le mot-clé `override` lors de la définition d'une méthode portant le même nom dans la sous-classe, puisque la méthode de la classe de base n'est pas visible à partir de la sous-classe.

  Pour redéfinir une méthode d'occurrence correspondant à ces critères, la définition de la méthode dans la sous-classe doit utiliser le mot-clé `override` et correspondre, comme défini ci-dessous, à la version de cette méthode dans la superclasse :
- La méthode de redéfinition doit avoir le même niveau de contrôle d'accès que celle de la classe de base. Les méthodes définies comme internes doivent avoir le même niveau de contrôle d'accès que celles qui n'ont pas de spécificateur de contrôle d'accès.
- La méthode de redéfinition doit avoir le même nombre de paramètres que celle de la classe de base.
- Les paramètres de la méthode de redéfinition doivent avoir les mêmes annotations de type de données que ceux de la méthode de la classe de base.
- La méthode de redéfinition doit avoir le même type de renvoi que celle de la classe de base.

Toutefois, il n'est pas nécessaire que les noms des paramètres de la méthode de redéfinition correspondent à ceux des paramètres de la classe de base, tant qu'il y a une correspondance entre le nombre de paramètres et leurs types de données.

Instruction super

Il arrive fréquemment que les programmeurs souhaitent compléter le comportement de la méthode de superclasse qu'ils redéfinissent, plutôt que de remplacer ce comportement. Il est donc nécessaire de disposer d'un mécanisme permettant à une méthode d'une sous-classe d'appeler sa propre superclasse. Ce mécanisme est assuré par l'instruction `super` qui contient une référence à la superclasse immédiate. L'exemple suivant définit une classe appelée Base, qui contient la méthode appelée `thanks()` et une sous-classe de Base appelée Extender, qui redéfinit la méthode `thanks()`. La méthode `Extender.thanks()` utilise l'instruction `super` pour appeler `Base.thanks()`.

```
package {
    import flash.display.MovieClip;
    public class SuperExample extends MovieClip
    {
        public function SuperExample()
        {
            var myExt:Extender = new Extender()
            trace(myExt.thanks()); // output: Mahalo nui loa
        }
    }
}

class Base {
    public function thanks():String
    {
        return "Mahalo";
    }
}

class Extender extends Base
{
    override public function thanks():String
    {
        return super.thanks() + " nui loa";
    }
}
```

Redéfinition des getters et setters

Bien qu'il soit impossible de redéfinir les variables définies dans une superclasse, il est possible de redéfinir les getters et setters. Par exemple, le code suivant redéfinit un getter appelé `currentLabel` qui est défini dans la classe MovieClip d'ActionScript 3.0 :

```
package
{
    import flash.display.MovieClip;
    public class OverrideExample extends MovieClip
    {
        public function OverrideExample()
        {
            trace(currentLabel)
        }
        override public function get currentLabel():String
        {
            var str:String = "Override: ";
            str += super.currentLabel;
            return str;
        }
    }
}
```

Le résultat de l'instruction `trace()` dans le constructeur de la classe OverrideExample est `Override: null`, ce qui montre que cet exemple a réussi à redéfinir la propriété héritée`currentLabel`.

Propriétés statiques non héritées

Les sous-classes n'héritent pas des propriétés statiques. Ces dernières ne sont donc pas accessibles par une occurrence d'une sous-classe. Une propriété statique n'est accessible que par le biais de l'objet classe dans lequel elle est définie. Par exemple, le code suivant définit une classe de base appelée Base et une sous-classe appelée Extender, qui étend la classe Base. Une variable statique appelée `test` est définie dans la classe Base. Le code de l'extrait ci-dessous ne peut être compilé en mode strict et génère une erreur d'exécution en mode standard.

```
package {
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
            trace(myExt.test);// error
        }
    }
}

class Base {
    public static var test:String = "static";
}

class Extender extends Base { }
```

La seule façon d'accéder à la variable statique `test` est par l'objet classe, comme le montre le code suivant :

```
Base.test;
```

Il est toutefois permis de définir une propriété d'occurrence ayant le même nom qu'une propriété statique. Cette propriété d'occurrence peut être définie dans la même classe que la propriété statique, ou dans une sous-classe. Par exemple, la classe Base de l'exemple précédent peut comporter une propriété d'occurrence appelée `test`. Le code suivant peut être compilé et exécuté normalement, car la classe Extender hérite de la propriété d'occurrence. Ce code peut aussi être compilé et exécuté normalement si la définition de la variable d'occurrence test est déplacée (mais non copiée) dans la classe Extender.

```
package
{
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
            trace(myExt.test);// output: instance
        }
    }
}

class Base
{
    public static var test:String = "static";
    public var test:String = "instance";
}

class Extender extends Base {}
```

Propriétés statiques et chaîne de portée

Bien que les propriétés statiques ne soient pas héritées, elles figurent dans la chaîne de portée de la classe qui les définit ainsi que de toute sous-classe de celle-ci. C'est pourquoi on dit que les propriétés statiques sont *dans la portée* à la fois de la classe dans laquelle elles sont définies et des sous-classes de celle-ci. Cela signifie qu'une propriété statique est directement accessible à partir du corps de la classe qui la définit et de toute sous-classe de celle-ci.

L'exemple suivant modifie les classes définies dans l'exemple précédent pour montrer que la variable statique `test`, définie dans la classe Base, est dans la portée de la classe Extender. Autrement dit, la classe Extender peut accéder à la variable statique `test` sans qu'il soit nécessaire de faire précéder le nom de celle-ci du nom de la classe qui définit `test`.

```
package
{
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
        }
    }
}

class Base {
    public static var test:String = "static";
}

class Extender extends Base
{
    public function Extender()
    {
        trace(test); // output: static
    }

}
```

Si une propriété d'occurrence est définie avec le même nom qu'une propriété statique de la même classe ou d'une superclasse, la propriété d'occurrence est prioritaire dans la chaîne de portée. On dit alors que la propriété d'occurrence *fait de l'ombre* à la propriété statique, ce qui signifie que la valeur de la propriété d'occurrence est utilisée à la place de celle de la propriété statique. Par exemple, le code ci-dessous montre que si la classe Extender définit une variable d'occurrence appelée `test`, l'instruction `trace()` utilise la valeur de la variable d'occurrence à la place de celle de la variable statique.

```
package
{
    import flash.display.MovieClip;
    public class StaticExample extends MovieClip
    {
        public function StaticExample()
        {
            var myExt:Extender = new Extender();
        }
    }
}

class Base
{
    public static var test:String = "static";
}

class Extender extends Base
{
    public var test:String = "instance";
    public function Extender()
    {
        trace(test); // output: instance
    }

}
```

Rubriques avancées

Cette section débute par un bref historique d'ActionScript et de la programmation orientée objets (POO). Elle se poursuit par une présentation du modèle d'objet d'ActionScript 3.0 et de la façon dont il permet à la nouvelle machine virtuelle d'ActionScript (AVM2, ActionScript Virtual Machine 2) d'être nettement plus rapide que les versions antérieures de Flash Player, basées sur la première version de la machine virtuelle d'ActionScript, AVM1.

Historique de la prise en charge de la programmation orientée objets en ActionScript

ActionScript 3.0 est une évolution des versions antérieures d'ActionScript. C'est pourquoi il peut être utile de comprendre l'évolution du modèle d'objet en ActionScript. ActionScript était à l'origine un simple mécanisme de script pour les premières versions de Flash. Les programmeurs Flash ont alors commencé à développer des applications de plus en plus complexes avec ActionScript. En réponse aux besoins de ces programmeurs, le langage de chaque nouvelle version a connu des ajouts destinés à faciliter la création d'applications complexes.

ActionScript 1.0

ActionScript 1.0 est le langage utilisé dans les versions 6 et antérieures de Flash Player. Même à ce stade précoce de développement, le modèle d'objet d'ActionScript était basé sur le concept de l'objet comme type fondamental de données. Un objet ActionScript est un type de données composite doté d'un groupe de *propriétés*. Dans le contexte d'un modèle d'objet, le terme *propriétés* désigne tout ce qui est affecté à un objet (variables, fonctions ou méthodes).

Bien que cette première génération d'ActionScript ne prenne pas en charge la définition de classes avec le mot-clé `class`, elle permet de définir une classe à l'aide d'un type spécial d'objet appelé objet prototype. Au lieu d'utiliser un mot-clé `class`pour créer une définition de classe abstraite qui sera ensuite instanciée en objets concrets (à l'instar des langages reposant sur des classes, comme Java ou C++), les langages basés sur le prototypage, tel ActionScript 1.0, utilisent un objet existant comme modèle (ou prototype) d'autres objets. Alors que les objets des langages basés sur la notion de classes peuvent pointer sur une classe qui leur sert de modèle, les objets des langages basés sur la notion de prototypes pointent sur un autre objet, leur prototype, qui leur sert de modèle.

Pour créer une classe en ActionScript 1.0, il est nécessaire de définir une fonction constructeur pour cette classe. En ActionScript, les fonctions sont des objets réels et non pas de simples définitions abstraites. La fonction constructeur sert alors d'objet prototype pour les occurrences de cette classe. Le code suivant crée une classe appelée Shape et définit une propriété appelée `visible` qui a la valeur `true` par défaut.

```
// base class
function Shape() {}
// Create a property named visible.
Shape.prototype.visible = true;
```

Cette fonction constructeur définit une classe Shape qui va être instanciée à l'aide de l'opérateur `new`, comme suit :

```
myShape = new Shape();
```

De même que l'objet de la fonction constructeur `Shape()` sert de prototype pour les occurrences de la classe Shape, il peut également servir pour les sous-classes de Shape, c'est-à-dire d'autres classes qui prolongent la classe Shape.

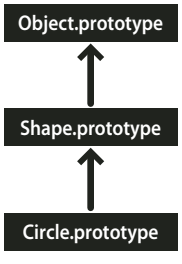
La création d'une classe qui est une sous-classe de la classe Shape est un processus en deux étapes. D'abord, créer la classe en définissant une fonction constructeur pour cette classe, comme suit :

```
// child class
function Circle(id, radius)
{
this.id = id;
this.radius = radius;
}
```

Ensuite, utiliser l'opérateur `new` pour déclarer que la classe Shape est le prototype de la classe Circle. Par défaut, toute nouvelle classe créée utilise la classe Object comme prototype, si bien que `Circle.prototype` contient alors un objet générique (une occurrence de la classe Object). Pour spécifier que le prototype de Circle est Shape et non pas Object, utilisez le code suivant pour changer la valeur de `Circle.prototype` afin qu'elle contienne un objet Shape et non plus un objet générique:

```
// Make Circle a subclass of Shape.
Circle.prototype = new Shape();
```

La classe Shape et la classe Circle sont maintenant liées par une relation d'héritage communément appelée *chaînage du prototype*. Le schéma représente la relation au sein d'un chaînage de prototype: chaînage de prototype :



La classe de base, à la fin de chaque chaînage de prototype, est la classe Object. La classe Object contient une propriété statique appelée `Object.prototype` qui pointe sur l'objet prototype de base pour tous les objets créés en ActionScript 1.0. Dans notre exemple de chaînage de prototype, l'objet suivant est l'objet Shape. En effet, la propriété `Shape.prototype` n'a jamais été explicitement définie, si bien qu'elle contient encore un objet générique (une occurrence de la classe Object). Le lien final de ce chaînage est la classe Circle qui est liée à son prototype, la classe Shape (la propriété `Circle.prototype` contient un objet Shape).

Si nous créons une occurrence de la classe Circle, comme dans l'exemple ci-dessous, l'occurrence hérite du chaînage de prototype de la classe Circle :

```
// Create an instance of the Circle class.
myCircle = new Circle();
```

Vous vous souvenez sans doute que nous avons créé une propriété appelée `visible` comme membre de la classe Shape. Dans notre exemple, la propriété `visible` n'existe pas comme partie de l'objet`myCircle`, mais uniquement comme membre de l'objet Shape ; et pourtant la ligne de code suivante produit la valeur `true` :

```
trace(myCircle.visible); // output: true
```

En remontant le chaînage du prototype, Flash Player est en mesure de vérifier que l'objet `myCircle` hérite de la propriété`visible`. Lorsque ce code est exécuté, Flash Player recherche d'abord dans les propriétés de l'objet `myCircle` une propriété appelée `visible`, mais ne la trouve pas. Flash Player recherche alors dans l'objet `Circle.prototype`, mais ne trouve toujours pas de propriété appelée `visible`. En continuant à remonter le chaînage du prototype, Flash Player trouve enfin la propriété `visible` définie dans l'objet `Shape.prototype` et affiche la valeur de cette propriété.

Par souci de simplicité, la présente section omet un grand nombre de détails et de subtilités du chaînage de prototype, puisqu'il s'agit simplement de vous aider à comprendre le modèle d'objet en ActionScript 3.0.

ActionScript 2.0

Avec ActionScript 2.0 ont été introduits de nouveaux mots-clés tels que `class`, `extends`, `public` et `private` qui permettaient de définir des classes selon une méthode bien connue de toute personne ayant travaillé avec les langages basés sur des classes, comme Java et C++. Il est important de comprendre que le mécanisme sous-jacent d'héritage n'a pas changé entre ActionScript 1.0 et ActionScript 2.0. Le passage à ActionScript 2.0 a consisté simplement à ajouter une nouvelle syntaxe pour la définition des classes. Le chaînage du prototype fonctionne de la même façon dans ces deux versions du langage.

La nouvelle syntaxe introduite par ActionScript 2.0 est représentée dans l'exemple ci-dessous. Elle permet de définir des classes d'une façon que la plupart des programmeurs considèrent comme plus intuitive :

```
// base class
class Shape
{
var visible:Boolean = true;
}
```

Vous pouvez remarquer qu'ActionScript 2.0 a également introduit les annotations de type destinées à une vérification des types à la compilation. Elles permettent de déclarer que la propriété `visible` de l'exemple précédent ne doit contenir qu'une valeur booléenne. Le nouveau mot-clé `extends` simplifie lui aussi le processus de création d'une sous-classe. Dans l'exemple suivant, ce qui nécessitait deux étapes en ActionScript 1.0 est accompli en une seule, à l'aide du mot-clé `extends` :

```
// child class
class Circle extends Shape
{
    var id:Number;
    var radius:Number;
    function Circle(id, radius)
    {
        this.id = id;
        this.radius = radius;
        }
}
```

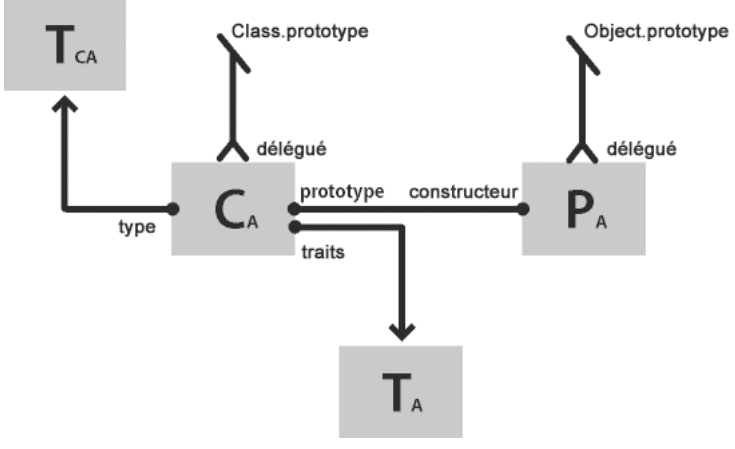
Le constructeur est maintenant déclaré dans le cadre de la définition de classe, et les propriétés `id` et `radius` de la classe doivent elles aussi être déclarées explicitement.

ActionScript 2.0 a également ajouté la prise en charge de la définition des interfaces, qui permet de rendre plus perfectionnés des programmes orientés objet, à l'aide de protocoles formellement définis pour la communication entre les objets.

Objet de classe en ActionScript 3.0

Un paradigme courant en programmation orientée objets, en particulier avec Java et C++, fait appel à des classes pour définir des types d'objets. Les langages de programmation qui adoptent ce paradigme ont aussi tendance à utiliser des classes pour construire des occurrences du type de données défini par la classe. ActionScript utilise des classes pour ces deux raisons, mais ses origines de langage basé sur des prototypes lui confèrent une caractéristique intéressante. Pour chaque définition de classe, ActionScript crée un objet de classe spécial qui autorise le partage du comportement et de l'état. Toutefois, pour de nombreux programmeurs en ActionScript, cette distinction n'aura aucune implication pratique sur le codage. En effet, ActionScript 3.0 est conçu de telle sorte qu'il est possible de créer des applications perfectionnées orientées objet sans utiliser, et sans même comprendre, ces objets de classe spéciaux. Cette section étudie en détail les enjeux pour les programmeurs avancés qui souhaitent tirer parti des objets de classe.

Le schéma suivant montre la structure d'un objet de classe représentant une classe simple appelée A, définie par l'instruction `class A`.



Chaque rectangle du schéma représente un objet. Chaque objet du schéma est marqué d'un caractère en indice pour signaler qu'il appartient à la classe A. L'objet de classe (CA) contient des références à un certain nombre d'autres objets importants. L'objet traits des occurrences (TA) enregistre les propriétés des occurrences qui sont définies dans une définition de classe. Un objet traits de la classe (TCA) représente le type interne de la classe et enregistre les propriétés statiques définies par la classe (le caractère C en indice signifie « classe »). L'objet prototype (PA) fait toujours référence à l'objet de classe auquel il était associé à l'origine par la propriété `constructeur`.

Objet traits

L'objet traits est une nouveauté d'ActionScript 3.0, implémentée pour des raisons de performances. Dans les versions précédentes d'ActionScript, la recherche d'un nom pouvait demander beaucoup de temps pendant que Flash Player remontait le chaînage du prototype. En ActionScript 3.0, la recherche d'un nom est beaucoup plus rapide et efficace, car les propriétés héritées sont copiées des superclasses dans l'objet traits des sous-classes.

L'objet traits n'est pas directement accessible par code, mais sa présence se manifeste par l'amélioration des performances et de l'utilisation de la mémoire. L'objet traits fournit à la machine virtuelle AVM2 des informations détaillées sur la disposition et le contenu d'une classe. Ces informations permettent à AVM2 de réduire nettement le temps d'exécution, car elle peut fréquemment générer des instructions machine directes pour accéder à des propriétés ou appeler des méthodes directement, sans effectuer au préalable une longue recherche de nom.

Grâce à l'objet traits, l'espace occupé en mémoire par un objet peut être nettement moins importante qu'avec les versions antérieures d'ActionScript. Par exemple, si une classe est scellée (c'est-à-dire si elle n'est pas déclarée comme dynamique), une occurrence de la classe ne nécessite pas de recherche par calcul d'adresse pour les propriétés ajoutées dynamiquement ; il lui suffit d'un pointeur sur l'objet traits et de quelques emplacements pour les propriétés fixes définies dans la classe. En conséquence, pour un objet qui nécessitait 100 octets en mémoire avec ActionScript 2.0, 20 suffiront avec ActionScript 3.0.

Remarque : *l'objet traits est un élément d'implémentation interne et il n'est pas garanti qu'il ne changera pas, ou même qu'il ne disparaîtra pas, dans les futures versions d'ActionScript.*

Objet prototype

En ActionScript, chaque objet de classe possède une propriété appelée `prototype`, qui est une référence à l'objet prototype de cette classe. L'objet prototype est un héritage des premières versions d'ActionScript, basées sur des prototypes. Pour plus d'informations, consultez la section « Historique de la prise en charge de la programmation orientée objets en ActionScript » à la page 119.

La propriété `prototype` est en lecture seule et ne peut donc pas être modifiée pour pointer sur d'autres objets. A l'inverse, dans les anciennes versions d'ActionScript, la propriété `prototype` pouvait être réaffectée pour pointer sur une autre classe. Bien que la propriété `prototype` soit en lecture seule, ce n'est pas le cas de l'objet prototype à laquelle elle fait référence. En d'autres termes, de nouvelles propriétés peuvent être ajoutées à l'objet prototype. Les propriétés ajoutées à l'objet prototype sont partagées avec toutes les autres occurrences de la classe.

Le chaînage de prototype, qui était le seul mécanisme d'héritage des versions antérieures d'ActionScript, n'a plus qu'un rôle secondaire en ActionScript 3.0. Le principal mécanisme d'héritage, l'héritage des propriétés fixes, est géré de façon interne par l'objet traits. Une propriété fixe est une variable ou une méthode définie dans le cadre d'une définition de classe. L'héritage des propriétés fixes est également appelé héritage des classes, car c'est le mécanisme d'héritage qui est associé aux mots-clés `class`, `extends` et `override`.

Le chaînage de prototype représente un autre mécanisme d'héritage qui est plus dynamique que l'héritage des propriétés fixes. Il est possible d'ajouter des propriétés à l'objet prototype d'une classe, non seulement dans le cadre de la définition de classe, mais aussi lors de l'exécution, par la propriété `prototype` de l'objet de classe. Notez toutefois que si le compilateur est en mode strict, il peut être impossible d'accéder aux propriétés ajoutées à un objet prototype, sauf si vous déclarez une classe avec le mot-clé `dynamic`.

La classe Object est un bon exemple d'une classe dont plusieurs propriétés sont attachées à l'objet prototype. Les méthodes `toString()` et `valueOf()` de la classe Object sont en fait des fonctions affectées à des propriétés de l'objet prototype de la classe Object. Voici un exemple de l'aspect théorique de la déclaration de ces méthodes (l'implémentation réelle est légèrement différente en raison des détails pratiques d'implémentation) :

```
public dynamic class Object
{
    prototype.toString = function()
    {
        // statements
    };
    prototype.valueOf = function()
    {
        // statements
    };
}
```

Comme nous l'avons déjà mentionné, il est possible d'attacher une propriété à l'objet prototype d'une classe en dehors de la définition de cette classe. Par exemple, la méthode`toString()` peut également être définie hors de la définition de la classe Object, comme suit :

```
Object.prototype.toString = function()
{
    // statements
};
```

Toutefois, contrairement à l'héritage des propriétés fixes, l'héritage du prototype ne nécessite pas le mot-clé `override` pour redéfinir une méthode de sous-classe. Par exemple, pour redéfinir la méthode `valueOf()` dans une sous-classe de la classe Object, trois options sont possibles. Premièrement, vous pouvez définir une méthode `valueOf()` dans l'objet prototype de la sous-classe, à l'intérieur de la définition de classe. Le code suivant crée une sous-classe d'Object appelée Foo et redéfinit la méthode `valueOf()` dans la définition de classe de l'objet prototype de Foo. Chaque classe héritant de la classe Object, il n'est pas nécessaire d'utiliser le mot-clé `extends`.

```
dynamic class Foo
{
    prototype.valueOf = function()
    {
        return "Instance of Foo";
    };
}
```

Deuxièmement, vous pouvez définir une méthode `valueOf()` dans l'objet prototype de Foo en-dehors de la définition de classe, comme le montre le code ci-dessous:

```
Foo.prototype.valueOf = function()
{
    return "Instance of Foo";
};
```

Troisièmement, vous pouvez définir une propriété fixe appelée `valueOf()` dans la classe Foo. Cette technique diffère des précédentes dans la mesure où elle mêle l'héritage des propriétés fixes à l'héritage du prototype. Pour redéfinir `valueOf()` à partir d'une sous-classe de Foo, il est nécessaire d'utiliser le mot-clé `override`. Le code ci-dessous montre `valueOf()` définie comme propriété fixe dans Foo :

```
class Foo
{
    function valueOf():String
    {
        return "Instance of Foo";
    }
}
```

Espace de noms d'ActionScript 3

L'existence de deux mécanismes d'héritage séparés, l'héritage des propriétés fixes et l'héritage du prototype, provoque un problème intéressant de compatibilité par rapport aux propriétés et méthodes des classe de base. La compatibilité avec la spécification du langage ECMAScript sur laquelle est basé ActionScript nécessite l'utilisation de l'héritage de prototype, si bien que les propriétés et les méthodes d'une classe de base sont définies dans l'objet prototype de cette classe. Mais par ailleurs, la compatibilité avec ActionScript 3.0 nécessite l'utilisation de l'héritage des propriétés fixes, si bien que les propriétés et méthodes d'une classe de base sont spécifiées dans la définition de classe, à l'aide des mots-clés `const`, `var` et `function`. De plus, l'utilisation des propriétés fixes plutôt que les versions du prototype est susceptible de permettre une nette amélioration des performances à l'exécution.

Pour résoudre ce problème, ActionScript 3.0 utilise les deux mécanismes d'héritage (propriétés fixes et prototype) pour les classes de base. Chaque classe de base contient deux jeux de propriétés et de méthodes. Un jeu est défini dans l'objet prototype pour assurer la compatibilité avec les spécifications d'ECMAScript et l'autre est défini avec les propriétés fixes et l'espace de noms d'AS3,0, pour assurer la compatibilité avec ActionScript 3.0.

L'espace de noms d'AS3 représente un mécanisme fort pratique pour choisir entre les deux jeux de propriétés et de méthodes. Si vous n'utilisez pas l'espace de noms d'AS3, une occurrence d'une classe de base hérite des propriétés et des méthodes définies dans l'objet prototype de cette classe de base. Si par contre vous utilisez l'espace de noms d'AS3, une occurrence d'une classe de base hérite des versions d'AS3, car les propriétés fixes sont toujours préférées aux propriétés du prototype. En d'autres termes, lorsqu'une propriété fixe est disponible, elle est toujours utilisée à la place d'une propriété du prototype ayant le même nom.

Il est possible d'utiliser sélectivement la version de l'espace de noms d'AS3 d'une propriété ou d'une méthode, en la qualifiant à l'aide de l'espace de noms d'AS3. Par exemple, le code ci-dessous utilise la version AS3 de la méthode `Array.pop()` :

```
var nums:Array = new Array(1, 2, 3);
nums.AS3::pop();
trace(nums); // output: 1,2
```

Vous pouvez aussi faire appel à la directive `use namespace` afin d'ouvrir l'espace de noms d'AS3 pour toutes les définitions figurant dans un bloc de code. Par exemple, le code ci-dessous utilise la directive `use namespace` afin d'ouvrir l'espace de noms d'AS3 pour les méthodes `pop()` et `push()`.

```
use namespace AS3;

var nums:Array = new Array(1, 2, 3);
nums.pop();
nums.push(5);
trace(nums) // output: 1,2,5
```

ActionScript 3.0 comporte aussi des options de compilation pour chaque jeu de propriétés, ce qui permet d'appliquer l'espace de noms d'AS3 au programme entier. L'option de compilation `-as3` représente l'espace de noms d'AS3 et l'option de compilation `-es` représente l'option d'héritage du prototype (`es` correspond à ECMAScript). Pour ouvrir l'espace de noms d'AS3 pour tout le programme, définissez l'option de compilation `-as3` sur `true` et l'option de compilation `-es` sur `false`. Pour utiliser les versions du prototype, définissez les options de compilation sur les valeurs opposées. Les options de compilation par défaut pour Adobe Flex Builder 3 et Adobe Flash CS4 Professional sont `-as3 = true` et `-es = false`.

Si vous prévoyez d'étendre l'une des classes de base et de redéfinir des méthodes, vous devez comprendre comment l'espace de noms d'AS3 affecte la façon de déclarer une méthode redéfinie. Si vous utilisez l'espace de noms d'AS3, toute redéfinition d'une méthode d'une classe de base doit également utiliser l'espace de noms d'AS3, ainsi que l'attribut `override`. Si vous n'utilisez pas l'espace de noms d'AS3 et voulez redéfinir une méthode d'une classe de base dans une sous-classe, vous ne devez utiliser ni l'espace de noms d'AS3 dans cette redéfinition, ni l'attribut `override`.

Exemple : GeometricShapes

L'exemple d'application GeometricShapes montre comment il est possible d'appliquer un certain nombre de concepts et fonctionnalités orientés objet à l'aide d'ActionScript 3.0 et, en particulier :

- Définition des classes
- Extension des classes
- Polymorphisme et le mot-clé `override`
- Définition et implémentation des interfaces

Cet exemple comprend aussi une « méthode usine (Factory) ».qui crée des occurrences de classes, montrant ainsi comment déclarer une valeur de retour comme occurrence d'une interface et utiliser l'objet ainsi renvoyé de manière générique.

Pour obtenir les fichiers de cet exemple d'application, consultez la page www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers de l'application GeometricShapes se trouvent dans le dossier Samples/GeometricShapes. L'application se compose des fichiers suivants :

Fichier	Description
GeometricShapes.mxml ou GeometricShapes.fla	Le fichier d'application principal dans Flash (FLA) ou Flex (MXML).
com/example/programmingas3/geometricshapes/IGeometricShape.as	Interface de base définissant les méthodes qui doivent être implémentées par toutes les classes de l'application GeometricShapes.
com/example/programmingas3/geometricshapes/IPolygon.as	Interface définissant les méthodes qui doivent être implémentées par les classes de l'application GeometricShapes qui comportent plusieurs côtés.
com/example/programmingas3/geometricshapes/RegularPolygon.as	Type de forme géométrique dont les côtés sont de longueur égale et positionnés symétriquement autour du centre de la forme.
com/example/programmingas3/geometricshapes/Circle.as	Type de forme géométrique qui définit un cercle.

Fichier	Description
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	Sous-classe de RegularPolygon qui définit un triangle équilatéral.
com/example/programmingas3/geometricshapes/Square.as	Sous-classe de RegularPolygon qui définit un carré.
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	Classe contenant une méthode usine (Factory) pour créer des formes à partir d'un type et d'une taille de forme.

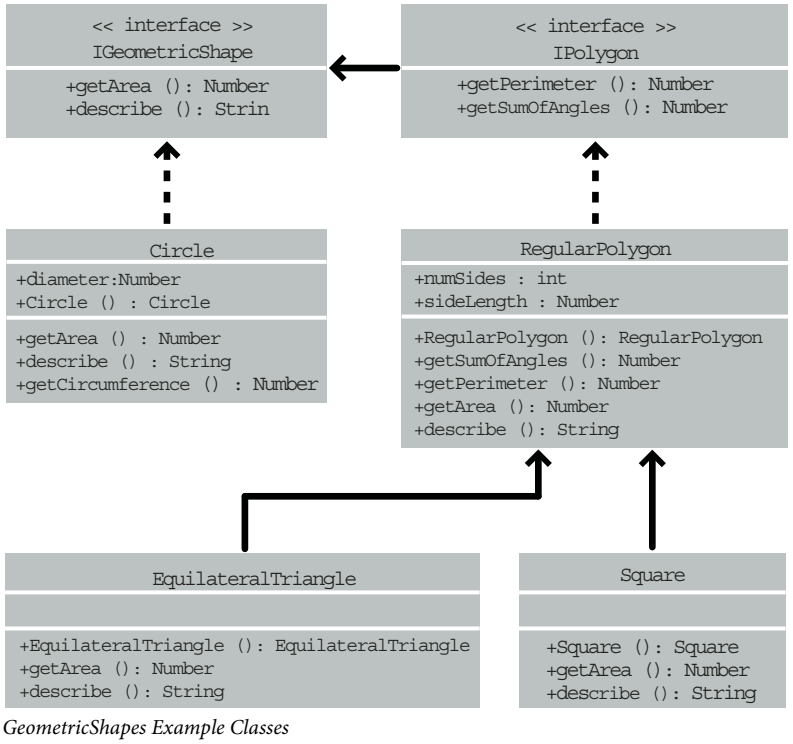
Définition des classes de GeometricShapes

L'application GeometricShapes permet de spécifier un type de forme géométrique et la taille de cette dernière. Elle renvoie alors la description de la forme, sa surface et son périmètre.

L'interface utilisateur de l'application est banale : elle se compose de quelques contrôles permettant de sélectionner le type de forme, de définir la taille et d'afficher la description. De fait, la partie intéressante de cette application est la face cachée de l'iceberg : la structure des classes et des interfaces elles-mêmes.

Cette application traite des formes géométriques, mais elle ne les affiche pas graphiquement. Elle représente une petite bibliothèque de classes et d'interfaces qui seront réutilisées dans un exemple, dans un autre chapitre (consultez la section « Exemple : SpriteArranger » à la page 322). L'exemple SpriteArranger affiche les formes graphiquement et permet à l'utilisateur de les manipuler, à partir de la structure de classes fournie ici dans l'application GeometricShapes.

Les classes et interfaces qui définissent les formes géométriques de cet exemple sont représentées dans le schéma ci-dessous, en notation UML (Unified Modeling Language) :



GeometricShapes Example Classes

Définition d'un comportement commun à l'aide d'interfaces

Cette application GeometricShapes gère trois types de formes: cercles, carrés et triangles équilatéraux. La structure des classes de GeometricShapes commence par une interface très simple, IGeometricShape, qui répertorie des méthodes communes aux trois types de formes :

```
package com.example.programmingas3.geometricshapes
{
    public interface IGeometricShape
    {
        function getArea():Number;
        function describe():String;
    }
}
```

L'interface définit deux méthodes : la méthode `getArea()` qui calcule et renvoie la surface de la forme, et la méthode `describe()` qui assemble une description textuelle des propriétés de la forme.

Nous voulons aussi connaître le périmètre de chaque forme. Toutefois, le périmètre d'un cercle est appelé circonférence et il est calculé de façon particulière, si bien que le comportement diverge de ceux d'un triangle ou d'un carré. Il existe cependant assez de similitudes entre les triangles, les carrés et les autres polygones pour qu'il soit logique de définir une nouvelle classe d'interface pour eux : IPolygon. L'interface IPolygon est également plutôt simple, comme on peut le constater :

```
package com.example.programmingas3.geometricshapes
{
    public interface IPolygon extends IGeometricShape
    {
        function getPerimeter():Number;
        function getSumOfAngles():Number;
    }
}
```

Cette interface définit deux méthodes communes à tous les polygones : la méthode `getPerimeter()` qui mesure la longueur combinée de tous les côtés et la méthode `getSumOfAngles()` qui additionne tous les angles intérieurs.

L'interface IPolygon étend l'interface IGeometricShape, si bien que toute classe qui implémente l'interface IPolygon doit déclarer les quatre méthodes : les deux de l'interface IGeometricShape et les deux de l'interface IPolygon.

Définition des classes de formes

Dès que vous avez une vue claire des méthodes communes à chaque type de forme, vous pouvez définir les classes de formes elles-mêmes. En ce qui concerne le nombre de méthodes à implémenter, la forme la plus simple est la classe Circle :

```
package com.example.programmingas3.geometricshapes
{
    public class Circle implements IGeometricShape
    {
        public var diameter:Number;

        public function Circle(diam:Number = 100):void
        {
            this.diameter = diam;
        }

        public function getArea():Number
        {
            // The formula is Pi * radius * radius.
            var radius:Number = diameter / 2;
            return Math.PI * radius * radius;
        }

        public function getCircumference():Number
        {
            // The formula is Pi * diameter.
            return Math.PI * diameter;
        }

        public function describe():String
        {
            var desc:String = "This shape is a Circle.\n";
            desc += "Its diameter is " + diameter + " pixels.\n";
            desc += "Its area is " + getArea() + ".\n";
            desc += "Its circumference is " + getCircumference() + ".\n";
            return desc;
        }
    }
}
```

La classe Circle implémente l'interface IGeometricShape. Elle doit donc comporter du code pour les méthodes `getArea()` et `describe()`. De plus, elle définit la méthode `getCircumference()` qui est unique à la classe Circle. La classe Circle déclare aussi une propriété, `diameter`, qui n'apparaîtra pas dans les autres classes, dédiées aux polygones.

Les deux autres types de formes, les carrés et les triangles équilatéraux, ont d'autres points communs : ils ont tous deux des côtés de longueur égale et il existe des formules communes pour calculer leurs périmètre et la somme de leurs angles intérieurs. En fait, ces formules communes s'appliquent à tout autre polygone régulier que vous êtes susceptible de définir par la suite.

La classe RegularPolygon sera la superclasse de la classe Square et de la classe EquilateralTriangle. Une superclasse permet de définir en un seul point des méthodes communes. Il n'est donc pas nécessaire de les définir séparément dans chaque sous-classe. Voici le code de la classe RegularPolygon :

```
package com.example.programmingas3.geometricshapes
{
    public class RegularPolygon implements IPolygon
    {
        public var numSides:int;
        public var sideLength:Number;

        public function RegularPolygon(len:Number = 100, sides:int = 3):void
        {
            this.sideLength = len;
            this.numSides = sides;
        }

        public function getArea():Number
        {
            // This method should be overridden in subclasses.
            return 0;
        }

        public function getPerimeter():Number
        {
            return sideLength * numSides;
        }

        public function getSumOfAngles():Number
        {
            if (numSides >= 3)
            {
                return ((numSides - 2) * 180);
            }
            else
            {
                return 0;
            }
        }

        public function describe():String
        {
            var desc:String = "Each side is " + sideLength + " pixels long.\n";
            desc += "Its area is " + getArea() + " pixels square.\n";
            desc += "Its perimeter is " + getPerimeter() + " pixels long.\n";
            desc += "The sum of all interior angles in this shape is " + getSumOfAngles() + "
degrees.\n";
            return desc;
        }
    }
}
```

La classe RegularPolygon déclare d'abord deux propriétés qui sont communes à tous les polygones réguliers : la longueur de chaque côté (la propriété `sideLength`) et le nombre de faces (la propriété `numSides`).

La classe RegularPolygon implémente l'interface IPolygon et déclare les quatre méthodes de l'interface IPolygon. Elle implémente deux d'entre elles, les méthodes `getPerimeter()` et `getSumOfAngles()` à l'aide de formules communes.

La formule de la méthode `getArea()` étant différente d'une forme à l'autre, la version de la méthode dans la classe de base ne peut pas comporter une logique commune dont hériteraient les méthodes des sous-classes. Elle renvoie donc simplement une valeur 0 par défaut pour indiquer que la surface n'a pas été calculée. Pour calculer correctement la surface de chaque forme, les sous-classes de la classe RegularPolygon devront redéfinir elles-mêmes la méthode `getArea()`.

Le code de la classe EquilateralTriangle, ci-dessous, montre comment la méthode `getArea()` :

```
package com.example.programmingas3.geometricshapes
{
    public class EquilateralTriangle extends RegularPolygon
    {
        public function EquilateralTriangle(len:Number = 100):void
        {
            super(len, 3);
        }

        public override function getArea():Number
        {
        // The formula is ((sideLength squared) * (square root of 3)) / 4.
        return ( (this.sideLength * this.sideLength) * Math.sqrt(3) ) / 4;
        }

        public override function describe():String
        {
                /* starts with the name of the shape, then delegates the rest
                of the description work to the RegularPolygon superclass */
        var desc:String = "This shape is an equilateral Triangle.\n";
        desc += super.describe();
        return desc;
        }
    }
}
```

Le mot-clé `override` indique que la méthode `EquilateralTriangle.getArea()` redéfinit volontairement la méthode `getArea()` de la superclasse RegularPolygon. Lorsque la méthode `EquilateralTriangle.getArea()` est appelée, elle calcule la surface à l'aide de la formule du code précédent. Le code de la méthode `RegularPolygon.getArea()` n'est jamais exécuté.

Par contre, la classe EquilateralTriangle ne définit pas sa propre version de la méthode`getPerimeter()`. Lorsque la méthode `EquilateralTriangle.getPerimeter()` est appelée, l'appel remonte la chaîne d'héritage et exécute le code de la méthode `getPerimeter()` de la superclasse RegularPolygon.

Le constructeur `EquilateralTriangle()` utilise l'instruction `super()` pour invoquer explicitement le constructeur `RegularPolygon()` de sa superclasse. Si les deux constructeurs avaient le même ensemble de paramètres, il serait possible d'omettre complètement le constructeur `EquilateralTriangle` et il suffirait d'exécuter le constructeur `RegularPolygon()`. Toutefois, le constructeur `RegularPolygon()` nécessite un paramètre supplémentaire, `numSides`. Le constructeur `EquilateralTriangle()` appelle donc `super(len, 3)` qui passe le paramètre en entrée `len` et la valeur 3 pour indiquer que le nombre de côtés du triangle est 3.

La méthode `describe()` utilise également l'instruction `super()`, mais de façon différente, afin d'invoquer la version de la méthode describe() `dans la superclasse RegularPolygon.` La méthode `EquilateralTriangle.describe()` définit d'abord la variable de chaîne `desc` qui affiche le type de forme. Elle obtient ensuite les résultats de la méthode `RegularPolygon.describe()` en appelant `super.describe()` et elle ajoute ces résultats à la fin de la chaîne `desc`.

La classe Square ne sera pas décrite en détail ici, mais elle est similaire à la classe EquilateralTriangle, avec un constructeur et ses propres implémentations des méthodes `getArea()` and `describe()`.

Polymorphisme et méthode usine (Factory)

Il est possible d'utiliser de diverses façons intéressantes un ensemble de classes qui utilisent à bon escient les interfaces et l'héritage. Par exemple, toutes les classes de formes décrites jusqu'ici implémentent l'interface IGeometricShape ou étendent une superclasse qui se charge de cette implémentation. Si l'on définit une variable comme étant une occurrence de IGeometricShape, il n'est donc pas nécessaire, pour appeler sa méthode `describe()`, de savoir si c'est une occurrence de la classe Circle ou de la classe Square.

Le code suivant illustre cette situation :

```
var myShape:IGeometricShape = new Circle(100);
trace(myShape.describe());
```

Lorsque `myShape.describe()` est appelée, elle exécute la méthode `Circle.describe()`; car bien que la variable soit définie comme une occurrence de l'interface IGeometricShape, Circle est sa classe sous-jacente.

Cet exemple illustre le principe du polymorphisme : le même appel à une méthode provoquera l'exécution d'un code différent, selon la classe de l'objet dont la méthode est appelée.

L'application GeometricShapes applique ce type de polymorphisme basé sur l'interface à l'aide d'une version simplifiée d'un modèle de conception appelé méthode usine (Factory). L'expression *méthode usine (Factory)* fait référence à une fonction qui renvoie un objet dont le type de données sous-jacent ou le contenu diffèrent selon le contexte.

La classe GeometricShapeFactory représentée ici définit une méthode usine (Factory) appelée `createShape()` :

```
package com.example.programmingas3.geometricshapes
{
    public class GeometricShapeFactory
    {
        public static var currentShape:IGeometricShape;

        public static function createShape(shapeName:String,

len:Number):IGeometricShape
        {
            switch (shapeName)
            {
                case "Triangle":
                    return new EquilateralTriangle(len);

                case "Square":
                    return new Square(len);

                case "Circle":
                    return new Circle(len);
            }
            return null;
        }

        public static function describeShape(shapeType:String, shapeSize:Number):String
        {
            GeometricShapeFactory.currentShape =
                GeometricShapeFactory.createShape(shapeType, shapeSize);
            return GeometricShapeFactory.currentShape.describe();
        }
    }
}
```

La méthode usine (Factory) `createShape()` laisse aux constructeurs de la sous-classe de la forme le soin de définir les détails des occurrences qu'ils créent, tout en renvoyant les nouveaux objets comme occurrences de IgeometricShape, ce qui permet à l'application de les traiter de façon plus générale.

La méthode `describeShape()` de l'exemple précédent montre comment une application peut utiliser la méthode usine (Factory) pour obtenir une référence générique à un objet spécifique. L'application peut obtenir la description d'un objet Circle nouvellement créé en procédant comme suit :

```
GeometricShapeFactory.describeShape("Circle", 100);
```

La méthode `describeShape()` appelle alors la méthode usine (Factory) `createShape()` avec les mêmes paramètres. Elle met le nouvel objet Circle dans une variable statique appelée `currentShape`, dont le type a été défini comme objet de IGeometricShape. La méthode `describe()` est ensuite appelée pour l'objet `currentShape` et cet appel est automatiquement résolu pour exécuter la méthode `Circle.describe()` qui renvoie une description détaillée du cercle.

Amélioration de l'application d'exemple

La puissance des interfaces et de l'héritage devient évidente dès qu'il s'agit de modifier l'application.

Supposons que nous voulions ajouter une nouvelle forme, un pentagone, à cette application. Il suffit de créer une nouvelle classe, Pentagon, qui étend la classe RegularPolygon et définit ses propres versions des méthodes `getArea()` et `describe()`. Une nouvelle option Pentagon est ensuite ajoutée dans l'interface utilisateur de l'application. C'est tout. La classe Pentagon obtiendra automatiquement, par héritage, les fonctionnalités des méthodes `getPerimeter()` et `getSumOfAngles()` de la classe RegularPolygon. Et puisqu'elle hérite d'une classe qui implémente l'interface IGeometricShape, une occurrence de Pentagon sera également traitée comme une occurrence de IGeometricShape. Autrement dit, il n'est pas nécessaire de modifier l'une des méthodes de la classe GeometricShapeFactory, ce qui rend beaucoup plus facile l'ajout ultérieur de nouveaux types de forme. Par conséquent, il n'est pas nécessaire non plus de modifier le code qui utilise la classe GeometricShapeFactory.

A titre d'exercice, il est conseillé d'ajouter une classe Pentagon à l'exemple Geometric Shapes, afin de constater à quel point les interfaces et l'héritage facilitent le processus d'ajout de nouvelles fonctionnalités à une application.

Chapitre 6 : Utilisation des dates et des heures

Si la ponctualité ne fait pas tout, elle n'en est pas moins un facteur clé des applications logicielles. ActionScript 3.0 propose différentes méthodes de gestion des dates calendaires, des heures et des intervalles temporels. Deux classes principales assurent l'essentiel de la fonctionnalité temporelle : la classe Date et la nouvelle classe Timer du package flash.utils.

Principes de base des dates et des heures

Introduction à l'utilisation des dates et des heures

Les dates et les heures constituent un type d'informations courant utilisé dans les programmes ActionScript. Par exemple, vous pouvez, entre autres, connaître la date du jour ou calculer combien de temps un utilisateur passe sur un écran particulier. Dans ActionScript, vous pouvez utiliser la classe Date pour représenter un moment unique dans le temps, y compris des informations de date et d'heure. Une occurrence de Date contient des valeurs pour les unités de date et d'heure individuelles (année, mois, date, jour de la semaine, heure, minutes, secondes, millisecondes et fuseau horaire, par exemple). Pour des utilisations plus avancées, ActionScript comprend également la classe Timer que vous pouvez utiliser pour effectuer des actions après un certain délai ou à des intervalles répétés.

Tâches de date et d'heure courantes

Ce chapitre décrit les tâches courantes suivantes qui utilisent des informations de date et d'heure :

- Utilisation des objets Date
- Obtention de la date et de l'heure actuelles
- Accès à des unités de date et d'heure individuelles (jours, années, heures, minutes, etc.)
- Exécution d'opérations arithmétiques sur les dates et les heures
- Conversion entre plusieurs fuseaux horaires
- Exécutions d'actions répétées
- Exécution d'actions après un délai défini

Concepts importants et terminologie

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Heure UTC : heure universelle, le fuseau horaire de référence d'heure zéro. Tous les autres fuseaux horaires sont définis sous la forme de nombre d'heures par rapport (avant ou après) à l'heure universelle.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester ses exemples de code. Etant donné que le code de ce chapitre traite essentiellement des objets Date, le test des exemples implique l'affichage des valeurs des variables utilisées dans les exemples, soit en les écrivant dans une occurrence de champ de texte sur la scène, soit en utilisant la fonction `trace()` pour imprimer des valeurs dans le panneau Sortie. Ces techniques sont décrites en détail à la section « Test des exemples de code contenus dans un chapitre » à la page 36.

Gestion des dates calendaires et des heures

Dans ActionScript 3.0, toutes les fonctions de gestion des dates calendaires et des heures se concentrent autour de la classe de niveau supérieur Date. Cette classe contient des méthodes et des propriétés qui vous permettent de manier les dates et les heures soit selon le modèle universel coordonné (UTC, Universal Time Coordinated), soit selon le fuseau horaire considéré. L'UTC est une définition normalisée du temps qui correspond essentiellement à l'heure du méridien de Greenwich (GMT, Greenwich Mean Time).

Création d'objets Date

La classe Date compte l'une des méthodes constructeur les plus polyvalentes de toutes les classes de base. Vous pouvez l'invoquer de quatre manières différentes.

Primo, si aucun paramètre n'est fourni, le constructeur `Date()` renvoie un objet Date contenant la date et l'heure locale actuelle de votre fuseau horaire. Exemple :

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

Secundo, si un paramètre numérique unique est fourni, le constructeur `Date()` le considère comme le nombre de millisecondes écoulées depuis le 1er janvier 1970 et renvoie l'objet Date correspondant. Notez que la valeur en millisecondes que vous transmettez est considérée comme le nombre de millisecondes depuis le 1er janvier 1970 en temps UTC. Toutefois, l'objet Date affiche des valeurs dans votre fuseau horaire local, sauf si vous utilisez des méthodes UTC uniquement pour les extraire et les afficher. Si vous créez un nouvel objet Date à l'aide du paramètre en millisecondes, veillez à tenir compte du décalage horaire entre votre fuseau local et le temps UTC. Les instructions ci-après créent un objet Date défini à minuit le 1er janvier 1970 en temps 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);
```

Tertio, vous pouvez transmettre plusieurs paramètres numériques au constructeur `Date()`. Celui-ci les traite respectivement comme la date, le mois, le jour, les heures, les minutes, les secondes et les millisecondes, et renvoie un objet Date correspondant. Les paramètres indiqués sont considérés comme correspondant à l'heure locale plutôt qu'au temps UTC. Les instructions suivantes établissent un objet Date défini à minuit le 1er janvier 2000, dans le fuseau horaire local :

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

Quarto, vous pouvez transmettre un paramètre numérique unique au constructeur `Date()`. Celui-ci essaiera d'analyser cette chaîne en composants de date et heure, et de renvoyer un objet Date correspondant. Si vous choisissez cette approche, il est recommandé d'inclure le constructeur `Date()` dans un bloc `try..catch` afin d'intercepter toute erreur d'analyse. Le constructeur `Date()` accepte plusieurs formats de chaînes, comme indiqué dans le Guide de référence du langage et des composants ActionScript 3.0. L'instruction suivante initialise un nouvel objet Date à l'aide d'une valeur chaîne :

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

Si le constructeur `Date()` ne peut analyser le paramètre chaîne, il ne déclenche pas d'exception. Cependant, l'objet Date résultant contiendra une valeur date incorrecte.

Obtention des valeurs d'unités temporelles

Vous pouvez extraire les valeurs de plusieurs unités temporelles au sein de l'objet Date grâce à des propriétés ou des méthodes de la classe Date. Chacune des propriétés suivantes fournit la valeur d'une unité temporelle de l'objet Date :

- La propriété `fullYear`
- La propriété `month` au format numérique : de 0 pour janvier à 11 pour décembre
- La propriété `date`, qui correspond au numéro calendaire du jour du mois, de 1 à 31
- La propriété `day`, qui correspond au jour de la semaine au format numérique, 0 représentant dimanche
- La propriété `hours`, de 0 à 23
- La propriété `minutes`
- La propriété `seconds`
- La propriété `milliseconds`

La classe Date offre en fait plusieurs manières d'obtenir ces valeurs. Vous pouvez par exemple obtenir la valeur month de l'objet Date de quatre manières :

- La propriété `month`
- La méthode `getMonth()`
- La propriété `monthUTC`
- La méthode `getMonthUTC()`

Ces quatre solutions sont d'une efficacité équivalente, vous pouvez donc adopter celle qui convient le mieux à votre application.

Les propriétés répertoriées ci-dessus représentent toutes des composants de la valeur date totale. Par exemple, la propriété milliseconds ne sera jamais supérieure à 999, puisque si elle atteint 1000, la valeur seconds augmente de 1, la propriété milliseconds se remet à zéro.

Si vous voulez obtenir la valeur de l'objet Date en millisecondes depuis le 1er janvier 1970 (UTC), vous pouvez utiliser la méthode `getTime()`. La méthode `setTime()`, quant à elle, vous permet de modifier la valeur d'un objet Date existant en millisecondes depuis le 1er janvier 1970 (UTC).

Opérations arithmétiques sur la date et l'heure

La classe Date permet d'additionner et de soustraire des dates et heures. Les valeurs Date sont conservées en interne sous forme de millisecondes. Il est donc nécessaire de convertir les autres valeurs en millisecondes avant de les ajouter ou de les soustraire aux objets Date.

Si votre application doit effectuer de nombreuses opérations arithmétiques sur les dates et heures, il peut s'avérer utile de créer des constantes qui conservent les valeurs d'unités temporelles courantes en millisecondes, comme illustré ci-après :

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

Il devient alors simple d'effectuer des opérations arithmétiques à l'aide des unités temporelles standard. Le code ci-après décale la valeur date d'une heure par rapport à l'heure actuelle à l'aide des méthodes `getTime()` et `setTime()` :

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

Une autre manière de définir une valeur date consiste à créer un objet Date à l'aide d'un seul paramètre en millisecondes. Par exemple, le code suivant ajoute 30 jours à une date pour en calculer une autre :

```
// 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));
```

La constante `millisecondsPerDay` est ensuite multipliée par 30 pour représenter la période de 30 jours et le résultat est ajouté à la valeur `invoiceDate` afin de définir la valeur `dueDate`.

Conversion entre plusieurs fuseaux horaires

Les opérations arithmétiques sur les dates et heures sont particulièrement pratiques lorsque vous devez convertir des dates d'un fuseau horaire à un autre. Tel est le rôle de la méthode `getTimezoneOffset()`, qui renvoie la valeur (en minutes) de décalage entre le fuseau horaire de l'objet Date et le temps UTC. La valeur renvoyée s'exprime en minutes parce que tous les fuseaux horaires ne correspondent pas à une heure ; certains sont décalés d'une demi-heure par rapport aux fuseaux voisins.

L'exemple suivant utilise le décalage de fuseau horaire pour convertir une date à partir de l'heure locale en temps UTC. La conversion s'effectue tout d'abord par calcul de la valeur du fuseau horaire en millisecondes, puis par ajustement de la valeur Date selon ce montant :

```
// 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);
```

Contrôle des intervalles temporels

Lorsque vous développez des applications à l'aide de Adobe Flash CS4 Professional, vous avez accès au scénario, qui fournit une progression constante, image par image, au sein de votre application. Toutefois, dans un projet purement ActionScript, vous devez compter sur d'autres mécanismes temporels.

Boucles ou minuteurs ?

Dans certains langages de programmation, vous devez mettre au point des motifs temporels à l'aide d'instructions en boucle telles que `for` ou `do..while`.

Les instructions en boucle s'exécutent en général aussi vite que la machine locale le permet, ce qui signifie que l'application sera plus rapide sur certaines machines que sur d'autres. Si votre application doit bénéficier d'un intervalle temporel cohérent, vous devez l'associer à un calendrier ou une horloge réels. Bien des applications, telles que les jeux, les animations, les contrôleurs en temps réel, nécessitent des mécanismes de décompte temporel qui soient cohérents d'une machine à l'autre.

La classe Timer d'ActionScript 3.0 offre une solution performante dans ce domaine. Grâce au modèle d'événements ActionScript 3.0, la classe Timer distribue des événements Timer dès qu'un intervalle spécifié est atteint.

La classe Timer

Pour la gestion des fonctions temporelles dans ActionScript 3.0, il est recommandé d'utiliser la classe Timer (flash.utils.Timer), qui permet de distribuer des événements dès qu'un intervalle est atteint.

Pour lancer un minuteur, vous devez d'abord créer une occurrence de la classe Timer et lui indiquer à quelle fréquence elle doit générer un événement Timer et combien de fois elle doit le faire avant de s'arrêter.

Par exemple, le code suivant crée une occurrence de Timer qui distribue un événement toutes les secondes pendant 60 secondes :

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

L'objet Timer distribue un objet TimerEvent chaque fois que l'intervalle donné est atteint. Le type d'événement de l'objet TimerEvent est `timer` (défini par la constante `TimerEvent.TIMER`). Un objet TimerEvent contient les mêmes propriétés que l'objet standard Event.

Si l'occurrence de Timer prévoit un nombre fixe d'intervalles, elle distribue également un événement `timerComplete` (défini par la constante `TimerEvent.TIMER_COMPLETE`) lorsqu'elle atteint l'intervalle final.

Voici un court exemple d'application illustrant la classe Timer en action :

```
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!");
        }
    }
}
```

Lorsque la classe ShortTimer est créée, elle génère une occurrence de Timer qui marque chaque seconde pendant cinq secondes. Elle ajoute alors deux écouteurs au minuteur : un qui écoute chaque décompte et un qui écoute l'événement `timerComplete`.

Elle lance ensuite le décompte du minuteur et à partir de là, la méthode `onTick()` s'exécute toutes les secondes.

La méthode `onTick()` affiche simplement le nombre actuel de tics. Après cinq secondes, la méthode `onTimerComplete()` s'exécute et vous avertit que le temps est écoulé.

Si vous exécutez cet exemple, vous devriez voir les lignes suivantes s'afficher dans votre console ou fenêtre de suivi à raison d'une ligne par seconde :

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

Fonctions temporelles du package flash.utils

ActionScript 3.0 contient un certain nombre de fonctions temporelles similaires à celles qui étaient disponibles dans ActionScript 2.0. Ces fonctions sont fournies au niveau du package dans le package flash.utils et elles fonctionnent de la même manière que dans ActionScript 2.0.

Fonction	Description
`clearInterval(id:uint):void`	Annule un appel de `setInterval()` spécifié.
`clearTimeout(id:uint):void`	Annule un appel de `setTimeout()` spécifié.
`getTimer():int`	Renvoie le nombre de millisecondes qui se sont écoulées depuis l'initialisation d'Adobe® Flash® Player ou d'Adobe® AIR™.
`setInterval(closure:Function, delay:Number, ... arguments):uint`	Exécute une fonction à fréquence définie (intervalle exprimé en millisecondes).
`setTimeout(closure:Function, delay:Number, ... arguments):uint`	Exécute une fonction spécifiée après un délai spécifié (en millisecondes).

Ces fonctions demeurent dans ActionScript 3.0 afin d'assurer la compatibilité avec les versions antérieures. Adobe ne recommande pas leur utilisation dans les nouvelles applications ActionScript 3.0. Il est en général plus simple et plus efficace d'utiliser la classe Timer.

Exemple : horloge analogique simple

Cet exemple d'horloge analogique simple illustre deux des concepts de date et heure étudiés dans ce chapitre :

- Obtention de la date et de l'heure actuelle et extraction des valeurs heures, minutes et secondes
- Utilisation d'une horloge pour fixer le rythme d'une application

  Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application SimpleClock se trouvent dans le dossier Samples/SimpleClock. L'application se compose des fichiers suivants :

Fichier	Description
SimpleClockApp.mxml ou SimpleClockApp.fla	Le fichier d'application principal dans Flash (FLA) ou Flex (MXML).
com/example/programmingas3/simpleclock/SimpleClock.as	Fichier d'application principal.
com/example/programmingas3/simpleclock/AnalogClockFace.as	Dessine un cadran d'horloge rond et les aiguilles des heures, des minutes et des secondes en fonction de l'heure.

Définition de la classe SimpleClock

Si l'exemple d'horloge est simple, il est toujours judicieux de bien organiser les applications de manière à faciliter leur extension future. Dans ce but, l'application SimpleClock utilise la classe SimpleClock pour gérer les tâches de démarrage et de mesure temporelle. Elle se sert ensuite d'une autre classe, AnalogClockFace, pour l'affichage réel de l'heure.

Voici le code qui définit et initialise la classe SimpleClock (vous remarquerez que dans la version Flash, SimpleClock étend la classe Sprite à la place) :

```
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;
```

Cette classe possède deux propriétés importantes :

- La propriété `face`, qui correspond à une occurrence de la classe AnalogClockFace
- La propriété `ticker`, qui est une occurrence de la classe Timer

  La classe SimpleClock utilise un constructeur par défaut. La méthode `initClock()` se charge de la véritable configuration, en créant le cadran et en lançant le décompte de l'occurrence Timer.

Création du cadran

Les lignes suivantes du code SimpleClock créent le cadran utilisé pour afficher l'heure :

```
    /**
     * 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();
```

La taille de l'horloge peut être transmise à la méthode `initClock()`. Si aucune valeur `faceSize` n'est transmise, la taille par défaut de 200 pixels est utilisée.

Ensuite, l'application initialise le cadran et l'ajoute à la liste d'affichage à l'aide de la méthode `addChild()` héritée de la classe DisplayObject. Elle appelle enfin la méthode `AnalogClockFace.draw()` pour afficher une fois le cadran, qui indique l'heure actuelle.

Lancement du minuteur

Une fois le cadran créé, la méthode `initClock()` définit le minuteur :

```
// 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();
```

Cette méthode commence par instancier une occurrence de Timer qui va distribuer un événement par seconde (toutes les 1 000 millisecondes). Comme le constructeur `Timer()` ne reçoit pas de second paramètre `repeatCount`, l'horloge se reproduit indéfiniment.

La méthode `SimpleClock.onTick()` s'exécute une fois par seconde après réception de l'événement `timer` :

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

La méthode `AnalogClockFace.draw()` dessine simplement le cadran de l'horloge et des aiguilles.

Affichage de l'heure actuelle

La plupart du code de la classe AnalogClockFace implique la définition des éléments d'affichage du cadran. Lors de son initialisation, AnalogClockFace dessine un contour circulaire, place des libellés numériques pour chaque heure, puis crée trois objets Shape, un pour l'aiguille des heures, un pour celle des minutes et un pour l'aiguille des secondes de l'horloge.

Lorsque l'application SimpleClock s'exécute, elle appelle la méthode `AnalogClockFace.draw()` toutes les secondes, comme suit :

```
/**
 * 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);
}
```

Cette méthode enregistre l'heure actuelle dans une variable, pour que l'heure ne puisse changer pendant le dessin des aiguilles de l'horloge. Elle appelle ensuite la méthode `showTime()` pour afficher les aiguilles, comme illustré ci-après :

```
/**
 * 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);
}
```

Tout d'abord, cette méthode extrait les valeurs des heures, des minutes et des secondes pour l'heure actuelle. Elle utilise ensuite ces valeurs pour calculer l'angle de chaque aiguille. Comme l'aiguille des secondes effectue une rotation complète en 60 secondes, elle tourne de 6 degrés par seconde (360/60). L'aiguille des minutes pivote selon le même angle chaque minute.

L'aiguille des heures se met à jour toutes les minutes également et doit donc progresser à chaque minute. Elle tourne de 30 degrés toutes les heures (360/12), mais pivote également d'un demi-degré toutes les minutes (30 degrés divisés par 60 minutes).

Chapitre 7 : Utilisation des chaînes

La classe String contient des méthodes qui vous permettent de manipuler des chaînes de texte. Les chaînes s'utilisent avec de nombreux objets. Les méthodes décrites dans ce chapitre sont utiles pour manipuler des chaînes dans des objets tels que TextField, StaticText, XML, ContextMenu et FileReference.

Les chaînes sont des séries de caractères. ActionScript 3.0 prend en charge les caractères ASCII et Unicode.

Principes de base des chaînes

Introduction à l'utilisation des chaînes

Pour les programmeurs, une « chaîne » est un texte, une suite de lettres, chiffres ou autres caractères qui se suivent et forment une unité représentée par une valeur. Par exemple, la ligne de code suivante crée une variable ayant le type de données String (chaîne) et affecte une valeur de chaîne littérale à cette variable :

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

Comme le montre cet exemple, ActionScript permet de délimiter une valeur chaîne en enfermant du texte entre des guillemets droits doubles ou simples. Voici d'autres exemples de chaînes :

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

Lorsque vous manipulez un fragment de texte en ActionScript, vous utilisez une valeur chaîne. La classe String d'ActionScript est le type de données qui permet de travailler avec du texte. Des occurrences de chaînes sont fréquemment utilisées pour des propriétés, des paramètres de méthodes, etc., dans de nombreuses autres classes en ActionScript.

Tâches courantes de manipulation des chaînes de texte

Voici quelques tâches courantes relatives aux chaînes de texte qui sont présentées dans ce chapitre :

- Création d'objets String
- Utilisation de caractères spéciaux (sauts de ligne, tabulations et caractères absents du clavier)
- Mesure de la longueur des chaînes
- Séparation de caractères individuels au sein d'une chaîne
- Concaténation de chaînes
- Comparaison de chaînes
- Recherche, extraction et remplacement de parties d'une chaîne
- Mise en majuscule ou minuscule de chaînes entières

Concepts importants et terminologie

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- ASCII : système de codage permettant de représenter du texte sous forme de caractères et symboles dans les programmes informatiques. Le système ASCII gère les 26 lettres de l'alphabet latin, plus un nombre limité de caractères supplémentaires.
- Caractère : unité de base d'un texte (lettre ou symbole).
- Concaténation : ajout bout à bout de plusieurs valeurs de chaîne pour en créer une nouvelle.
- Chaîne vide : chaîne qui ne contient rien: ni texte, ni espace, ni autre caractère, représentée par `""`. Une chaîne vide n'est pas la même chose qu'une variable ayant une valeur nulle (null) : celle-ci est une variable à laquelle aucune occurrence de l'objet String n'est affectée, alors qu'une chaîne vide est une occurrence dont la valeur ne contient aucun caractère.
- Chaîne : valeur textuelle (séquence de caractères).
- Littéral (ou « littéral de chaîne ») : valeur chaîne écrite explicitement en code, sous forme d'une valeur texte encadrée par des guillemets droits simples ou doubles.
- Sous-chaîne : définit une chaîne qui est une partie d'une autre chaîne.
- Unicode : système standardisé de codage permettant de représenter du texte sous forme de caractères et symboles dans les programmes informatiques. Le système Unicode permet d'utiliser la totalité des caractères de toutes les langues écrites existantes.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans le chapitre, vous pouvez tester des exemples de code. Etant donné que le code de ce chapitre traite essentiellement de la manipulation de texte, le test des exemples implique l'affichage des valeurs des variables utilisées dans les exemples, soit en les écrivant dans une occurrence de champ de texte sur la scène, soit en utilisant la fonction `trace()` pour imprimer des valeurs dans le panneau Sortie. Ces techniques sont décrites en détail dans « Test des exemples de code contenus dans un chapitre » à la page 36.

Création de chaînes

La classe String permet de représenter des données de chaîne (texte) en ActionScript 3.0. Les chaînes ActionScript prennent en charge les caractères ASCII et Unicode. La meilleure façon de créer une chaîne est d'utiliser un littéral de chaîne. Pour déclarer un littéral de chaîne, utilisez les guillemets droits doubles (`"`) ou les guillemets droits simples (`'`). Par exemple, les deux chaînes suivantes sont équivalentes :

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

Vous pouvez également déclarer une chaîne à l'aide de l'opérateur `new`, comme suit :

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

Les deux chaînes suivantes sont équivalentes :

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

Pour utiliser des guillemets droits simples (`'`) dans un littéral de chaîne délimité par des guillemets droits simples (`'`), utilisez le caractère d'échappement (`\`). De même, pour utiliser des guillemets droits doubles (`''`) dans un littéral de chaîne délimité par des guillemets droits doubles (`''`), utilisez le caractère d'échappement (`\`). Les deux chaînes suivantes sont équivalentes :

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

Vous pouvez utiliser des guillemets simples ou des guillemets doubles en fonction de ceux qui existent dans un littéral de chaîne, comme dans l'exemple suivant :

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

Rappelons qu'ActionScript fait la distinction entre les guillemets droits simples (`'`) et les guillemets simples gauches ou droits (`'` ou `'`). Il en est de même pour les guillemets doubles. Utilisez des guillemets droits pour délimiter des littéraux de chaîne. Lorsque vous collez du texte dans ActionScript depuis une autre source, utilisez les caractères corrects.

Comme indiqué dans le tableau suivant, vous pouvez utiliser le caractère d'échappement (`\`) pour définir d'autres caractères dans des littéraux de chaîne :

Séquence d'échappement	**Caractère**
`\b`	Retour arrière
`\f`	Changement de page
`\n`	Nouvelle ligne
`\r`	Retour chariot
`\t`	Tab
`\u`***nnnn***	Caractère Unicode dont le code de caractère est spécifié par le nombre hexadécimal *nnnn* `; par exemple, \u263a` est le caractère smiley.
`\\x`***nn***	Caractère ASCII dont le code est spécifié par le nombre hexadécimal *nn*.
`\'`	Guillemet droit simple
`\"`	Guillemet droit double
`\\`	Barre oblique inverse

Propriété length

Chaque chaîne possède une propriété `length` correspondant au nombre de caractères qu'elle contient:

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

Une chaîne vide et une chaîne null ont toutes deux une longueur de 0, comme l'indique l'exemple suivant :

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

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

Utilisation de caractères dans des chaînes

Chaque caractère d'une chaîne possède une position d'index dans la chaîne (un entier). La position d'index du premier caractère est 0. Par exemple, dans la chaîne suivante, le caractère `y` occupe la position 0 et le caractère `w` occupe la position 5 :

```
"yellow"
```

Vous pouvez examiner des caractères individuels à différentes positions d'une chaîne à l'aide des méthodes `charAt()` et `charCodeAt()`, comme dans l'exemple suivant :

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

Lorsque vous exécutez ce code, vous obtenez le résultat suivant :

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

Vous pouvez également utiliser des codes de caractère pour définir une chaîne à l'aide de la méthode `fromCharCode()`, comme l'indique l'exemple suivant :

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

Comparaison de chaînes

Vous pouvez utiliser les opérateurs suivants pour comparer des chaînes : `<`, `<=`, `!=`, `==`, `=>` et `>`. Vous pouvez utiliser ces opérateurs avec des instructions conditionnelles (`if` et `while`, par exemple) comme l'indique l'exemple suivant :

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

Lorsque vous utilisez ces opérateurs avec des chaînes, ActionScript considère la valeur du code de chaque caractère dans la chaîne, en comparant les caractères de gauche à droite, comme indiqué ci-dessous :

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

Utilisez les opérateurs `==` et `!=` pour comparer des chaînes entre elles et pour comparer des chaînes avec d'autres types d'objets, comme l'indique l'exemple suivant :

```
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
```

Récupération des représentations de chaîne d'autres objets

Vous pouvez obtenir une représentation de chaîne de n'importe quel type d'objet. Tous les objets disposent en effet d'une méthode `toString()` :

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

Lorsque vous utilisez l'opérateur de concaténation `+` avec une combinaison d'objet String et d'objets qui ne sont pas des chaînes, vous n'avez pas besoin d'utiliser la méthode `toString()`. Pour plus d'informations sur la concaténation, consultez la section suivante.

La fonction globale `String()` renvoie la même valeur pour un objet donné que la valeur renvoyée par l'objet appelant la méthode `toString()`.

Concaténation de chaînes

La concaténation de chaînes consiste à prendre deux chaînes et à les combiner en une seule chaîne de façon séquentielle. Par exemple, vous pouvez utiliser l'opérateur `+` pour concaténer deux chaînes :

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

Vous pouvez également utiliser l'opérateur `+=` pour obtenir le même résultat, comme dans l'exemple suivant :

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

En outre, la classe String comprend la méthode `concat()`, utilisable comme suit :

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

Si vous utilisez l'opérateur + (ou l'opérateur +=) avec un objet String et un objet qui n'est *pas* une chaîne, ActionScript convertit automatiquement ce dernier en un objet String afin d'évaluer l'expression, comme indiqué dans l'exemple :

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

Néanmoins, vous pouvez utiliser des parenthèses pour le regroupement afin de fournir un contexte à l'opérateur +, comme indiqué dans l'exemple suivant :

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

Recherche de sous-chaînes et de modèles dans des chaînes

Les sous-chaînes sont des caractères consécutifs à l'intérieur d'une chaîne. La chaîne `"abc"`, par exemple, contient les sous-chaînes suivantes : `""`, `"a"`, `"ab"`, `"abc"`, `"b"`, `"bc"`, `"c"`. Vous pouvez utiliser des méthodes ActionScript pour localiser les sous-chaînes d'une chaîne.

Les modèles sont définis en ActionScript par des chaînes ou par des expressions régulières. Par exemple, l'expression régulière suivante définit un modèle spécifique, les lettres A, B, et C suivies d'un chiffre (les barres de fraction sont des délimiteurs d'expression régulière) :

```
/ABC\d/
```

ActionScript comporte des méthodes servant à rechercher des modèles dans des chaînes et à remplacer les correspondances trouvées par des sous-chaînes de substitution. Ces méthodes sont décrites dans les sections suivantes.

Les expressions régulières peuvent définir des modèles compliqués. Pour plus d'informations, consultez le chapitre « Utilisation d'expressions régulières » à la page 211.

Recherche d'une sous-chaîne par la position d'un caractère

Les méthodes `substr()` et `substring()` sont similaires. Elles renvoient toutes les deux une sous-chaîne d'une chaîne. Elles prennent deux paramètres. Dans les deux méthodes, le premier paramètre est la position du caractère initial dans la chaîne concernée. Toutefois, dans la méthode `substr()`, le deuxième paramètre est la *longueur* de la sous-chaîne à renvoyer, alors que dans la méthode `substring()`, le deuxième paramètre est la position du caractère *final* de la sous-chaîne (qui ne figure pas dans la chaîne renvoyée). Cet exemple illustre la différence entre ces deux méthodes :

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

La méthode `slice()` fonctionne de la même façon que la méthode `substring()`. Lorsque deux nombres entiers non négatifs sont passés en paramètres, son fonctionnement est identique. Néanmoins, la méthode `slice()` peut recevoir des entiers négatifs comme paramètres. Dans ce cas, la position des caractères est comptée à partir de la fin de la chaîne, comme dans l'exemple suivant :

```
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
```

Vous pouvez associer des entiers positifs et négatifs comme paramètres de la méthode `slice()`.

Recherche de la position des caractères d'une sous-chaîne correspondante

Vous pouvez utiliser les méthodes `indexOf()` et `lastIndexOf()` pour localiser des sous-chaînes correspondantes au sein d'une chaîne, comme dans l'exemple suivant :

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

La méthode `indexOf()` est sensible à la casse.

Vous pouvez spécifier un deuxième paramètre pour indiquer la position de l'index dans la chaîne à partir de laquelle commencer la recherche, comme suit :

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

La méthode `lastIndexOf()` trouve la dernière occurrence d'une sous-chaîne dans la chaîne :

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

Si vous incluez un deuxième paramètre avec la méthode `lastIndexOf()`, la recherche est effectuée à l'envers à partir de cette position d'index dans la chaîne (de droite à gauche) :

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

Création d'un tableau de sous-chaînes segmenté par un délimiteur

Vous pouvez utiliser la méthode `split()` pour créer un tableau de sous-chaînes divisé en fonction d'un caractère délimiteur. Par exemple, vous pouvez segmenter une chaîne séparée par des virgules ou des tabulations en plusieurs chaînes.

L'exemple suivant indique comment diviser un tableau en sous-chaînes avec le caractère esperluette (&) comme délimiteur :

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

Le deuxième paramètre de la méthode `split()` (qui est facultatif) définit la taille maximale du tableau renvoyé.

Vous pouvez également utiliser une expression régulière comme caractère délimiteur:

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

Pour plus d'informations, consultez le chapitre « Utilisation d'expressions régulières » à la page 211 et le Guide de référence du langage et des composants ActionScript 3.0.

Recherche de modèles dans des chaînes et remplacement de sous-chaînes

La classe String comprend les méthodes suivantes pour utiliser des modèles dans des chaînes :

- Utilisez les méthodes `match()` et `search()` pour localiser des sous-chaînes qui correspondent à un modèle.
- Utilisez la méthode `replace()` pour rechercher des sous-chaînes qui correspondent à un modèle et les remplacer par une sous-chaîne spécifiée.

Ces méthodes sont décrites dans les sections suivantes.

Vous pouvez utiliser des chaînes ou des expressions régulières pour définir des modèles utilisés dans ces méthodes. Pour plus d'informations sur les expressions régulières, consultez le chapitre « Utilisation d'expressions régulières » à la page 211.

Recherche de sous-chaînes correspondantes

La méthode `search()` renvoie la position de l'index de la première sous-chaîne qui correspond à un modèle donné, comme illustré dans cet exemple :

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

Vous pouvez également utiliser des expressions régulières pour définir le modèle pour lequel établir une correspondance, comme illustré dans cet exemple :

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

Le résultat de la méthode `trace()` est 0, car le premier caractère dans la chaîne est la position de l'index 0. L'indicateur `i` est défini dans l'expression régulière. Par conséquent, la recherche n'est pas sensible à la casse.

La méthode `search()` trouve une seule correspondance et renvoie sa position d'index de début, même si l'indicateur `g` (global) est défini dans l'expression régulière.

L'exemple suivant présente une expression régulière plus compliquée qui correspond à une chaîne dans des guillemets doubles :

```
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.)
```

La méthode `match()` fonctionne de façon similaire. Elle recherche une sous-chaîne correspondante. Néanmoins, lorsque vous utilisez l'indicateur global dans un modèle d'expression régulière (comme dans l'exemple suivant), `match()` renvoie un tableau de sous-chaînes correspondantes :

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

Le tableau `results` est défini comme suit :

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

Pour plus d'informations sur les expressions régulières, consultez le chapitre « Utilisation d'expressions régulières » à la page 211« Utilisation d'expressions régulières » à la page 211.

Remplacement de sous-chaînes mises en correspondance

Vous pouvez utiliser la méthode `replace()` pour rechercher un modèle spécifié dans une chaîne et remplacer les correspondances par la chaîne de remplacement spécifiée, comme illustré dans l'exemple suivant :

```
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.
```

Dans cet exemple, les chaînes mises en correspondance ne sont pas sensibles à la casse car l'indicateur `i` (`ignoreCase`) est défini dans l'expression régulière, et plusieurs correspondances sont remplacées car l'indicateur `g` (`global`) est défini. Pour plus d'informations, consultez le chapitre « Utilisation d'expressions régulières » à la page 211.

Vous pouvez inclure les codes de remplacement `$` suivants dans la chaîne de remplacement. Le texte de remplacement indiqué dans le tableau suivant est inséré à la place du code de remplacement `$` :

Code $	Texte de remplacement
`$$`	$
`$&`	Sous-chaîne correspondante.
`$` `	Partie de la chaîne qui précède la sous-chaîne correspondante. Ce code utilise les guillemets simples droits gauches (`) et non les guillemets simples droits (') ni les guillemets simples anglais gauches (').
`$'`	Partie de la chaîne qui suit la sous-chaîne correspondante. Ce code utilise les guillemets simples droits (').
$n	*n*ième groupe entre parenthèses correspondant capturé, où n est un chiffre compris entre 1 et 9 et $n n'est pas suivi d'une décimale.
$nn	*nn*ième groupe entre parenthèses correspondant capturé, où *nn* est un nombre décimal à deux chiffres compris entre 01 et 99. Si la *nn*ième capture n'est pas définie, le texte de remplacement est une chaîne vide.

L'exemple suivant illustre l'utilisation des codes de remplacement `$2` et `$1`, qui représentent le premier et le deuxième groupes capturés correspondants :

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

Vous pouvez également utiliser une fonction comme deuxième paramètre de la méthode `replace()`. Le texte correspondant est remplacé par la valeur renvoyée de la fonction.

```
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;
}
```

Lorsque vous utilisez une fonction comme deuxième paramètre de la méthode `replace()`, les arguments suivants sont transmis à la fonction :

- La partie correspondante de la chaîne.
- Tout groupe entre parenthèses capturé correspondant. Le nombre d'arguments transmis de cette façon varie selon le nombre de correspondances entre parenthèses. Pour déterminer le nombre de correspondances entre parenthèses, vérifiez `arguments.length - 3` dans le code de la fonction.
- La position d'index dans la chaîne où débute la correspondance.
- La chaîne complète.

Conversion de la casse dans des chaînes

Comme illustré dans l'exemple suivant, les méthodes `toLowerCase()` et `toUpperCase()` convertissent les caractères alphabétiques de la chaîne en minuscule et en majuscule, respectivement:

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

Une fois que ces méthodes sont exécutées, la chaîne source reste inchangée. Pour transformer la chaîne source, utilisez le code suivant :

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

Ces méthodes fonctionnent avec des caractères étendus, pas simplement a–z et A–Z:

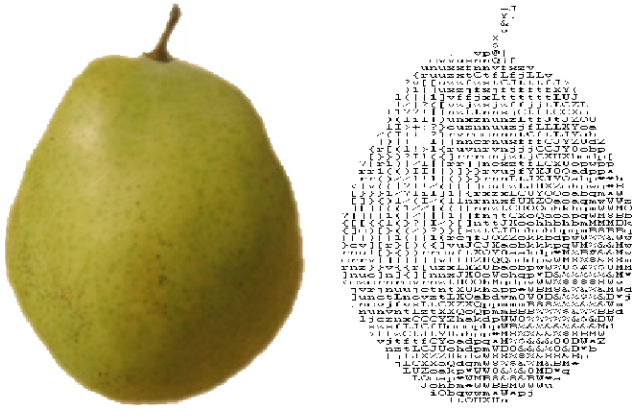
```
var str:String = "José Barça";
trace(str.toUpperCase(), str.toLowerCase()); // JOSÉ BARÇA josé barça
```

Exemple : ASCII art

Cet exemple ASCII Art représente différentes façons d'utiliser la classe String en ActionScript 3.0, notamment :

- La méthode `split()` de la classe String est utilisée pour extraire des valeurs d'une chaîne séparée par des caractères (informations d'image dans un fichier de texte séparé par des tabulations).
- Plusieurs techniques de manipulation de chaînes, y compris `split()`, la concaténation et l'extraction d'une partie de la chaîne à l'aide de `substring()` et de `substr()`, sont utilisées pour mettre la première lettre de chaque mot dans les titres d'image en majuscule.
- La méthode `getCharAt()` est utilisée pour obtenir un seul caractère à partir d'une chaîne (pour déterminer le caractère ASCII correspondant à une valeur bitmap d'échelle de gris).
- La concaténation de chaîne est utilisée pour construire la représentation ASCII art d'une image, un caractère à la fois.

Le terme *ASCII art* fait référence à des représentations textuelles d'une image dans lesquelles une grille de polices de caractères à espacement constant (caractères Courier New, par exemple) trace l'image. L'image suivante est un exemple d'ASCII art produit par l'application :



La version ASCII art du graphique est illustrée à droite.

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers de l'application ASCIIArt se trouvent dans le dossier Samples/AsciiArt. L'application se compose des fichiers suivants :

Fichier	Description
AsciiArtApp.mxml ou AsciiArtApp.fla	Fichier d'application principal en FLA pour Flash ou en MXML pour Flex
com/example/programmingas3/asciiArt/AsciiArtBuilder.as	La classe qui fournit la fonctionnalité principale de l'application, notamment l'extraction de métadonnées d'image d'un fichier de texte, le chargement des images et la gestion du processus de conversion d'image en texte.
com/example/programmingas3/asciiArt/BitmapToAsciiConverter.as	Une classe qui fournit la méthode `parseBitmapData()` pour convertir des données d'image dans une version String.
com/example/programmingas3/asciiArt/Image.as	Une classe qui représente une image bitmap chargée.
com/example/programmingas3/asciiArt/ImageInfo.as	Une classe représentant des métadonnées pour une image ASCII art (titre, URL de fichier image, etc.).
image/	Un dossier contenant des images utilisées par l'application.
txt/ImageData.txt	Un fichier de texte séparé par des tabulations et contenant des informations sur les images à charger par l'application.

Extraction de valeurs séparées par des tabulations

Cet exemple utilise le stockage séparé de données d'application par rapport à l'application ; de cette façon, si les données changent (par exemple, si une autre image est ajoutée ou que le titre d'une image change), il est inutile de recréer le fichier SWF. Dans ce cas, les métadonnées d'image, y compris le titre de l'image, l'URL du fichier d'image réel et certaines valeurs utilisées pour manipuler l'image, sont stockés dans un fichier de texte (le fichier txt/ImageData.txt dans le projet). Le contenu du fichier de texte est le suivant :

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

Le fichier utilise un format séparé par des tabulations spécifique. La première ligne est une ligne d'en-tête. Les lignes restantes contiennent les données suivantes pour chaque bitmap à charger :

- Le nom de fichier du bitmap.
- Le nom d'affichage du bitmap.
- Les valeurs de seuil du blanc et les valeurs de seuil du noir pour les bitmaps. Il s'agit de valeurs hexadécimales au-dessus et en dessous desquelles un pixel doit être considéré comme totalement blanc ou totalement noir.

Dès que l'application démarre, la classe AsciiArtBuilder se charge et analyse le contenu du fichier de texte afin de créer la « `pile` » d'images qu'elle chargera. Pour cela, elle utilise le code suivant de la méthode parseImageInfo() de la classe 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);
    }
}
```

Le contenu entier du fichier de texte se trouve dans une seule occurrence de String, la propriété `_imageInfoLoader.data`. Vous pouvez utiliser la méthode `split()` avec le caractère de nouvelle ligne (`"\n"`) comme paramètre pour diviser l'occurrence de String en un tableau (`lines`) dont les éléments sont les lignes individuelles du fichier de texte. Le code utilise ensuite une boucle pour travailler avec chacune des lignes (excepté la première car elle contient uniquement des en-têtes). A l'intérieur de la boucle, la méthode `split()` est de nouveau utilisée pour diviser le contenu de la ligne en un ensemble de valeurs (l'objet Array appelé `imageProperties`). Le paramètre utilisé avec la méthode `split()` dans ce cas est le caractère de tabulation (`"\t"`) car les valeurs dans chaque ligne sont délimitées par des tabulations.

Utilisation de méthodes String pour normaliser des titres d'image

Dans cette application, tous les titres d'image sont affichés à l'aide d'un format standard, avec la première lettre de chaque mot en majuscule (excepté quelques mots qui ne sont généralement pas en majuscule dans des titres anglais). Au lieu de considérer que le fichier de texte contient des titres formatés correctement, l'application formate les titres lors de leur extraction du fichier de texte.

Dans la liste de code précédente, lors de l'extraction de valeurs de métadonnées d'image individuelles, la ligne de code suivante est utilisée :

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

Dans ce code, le titre de l'image issu du fichier de texte est transmis au moyen de la méthode `normalizeTitle()` avant d'être stocké dans l'objet 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(" ");
}
```

Cette méthode utilise la méthode `split()` pour diviser le titre en mots individuels (séparés par le caractère d'espacement), transmet chaque mot au moyen de la méthode `capitalizeFirstLetter()` puis utilise la méthode `join()` de la classe Array pour combiner de nouveau les mots en une chaîne unique.

Comme son nom l'indique, la méthode `capitalizeFirstLetter()` met la première lettre de chaque mot en majuscule :

```
/**
 * 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;
}
```

En anglais, le premier caractère des mots suivants utilisés dans un titre n'est *pas* mis en majuscule : “and,” “the,” “in,” “an,” “or,” “at,” “of,” ou “a.” (Il s'agit d'une version simplifiée des règles.) Pour exécuter cette logique, le code utilise d'abord une instruction `switch` pour vérifier si le mot est l'un des mots ne devant pas être en majuscule. Si c'est le cas, le code sort de l'instruction `switch`. Si le mot doit être en majuscule, la procédure comprend plusieurs étapes :

1. La première lettre du mot est extraite à l'aide de `substr(0, 1)`, qui extraie une sous-chaîne commençant par le caractère au niveau de l'index 0 (la première lettre de la chaîne, comme indiqué par le premier paramètre `0`). La sous-chaîne contiendra un caractère (indiqué par le deuxième paramètre `1`).
2. Ce caractère est mis en majuscule à l'aide de la méthode `toUpperCase()`.

3 Les caractères restants du mot d'origine sont extraits à l'aide de `substring(1)`, qui extraie une sous-chaîne commençant à l'index 1 (la deuxième lettre) jusqu'à la fin de la chaîne (indiqué en omettant le deuxième paramètre de la méthode `substring()`).

4 Le dernier mot est créé en combinant la première lettre mise en majuscule et les lettres restantes en utilisant la concaténation de chaîne : `firstLetter + otherLetters`

Génération du texte ASCII art

La classe BitmapToAsciiConverter permet de convertir une image bitmap en sa représentation de texte ASCII. Cette procédure est effectuée par la méthode `parseBitmapData()`, partiellement représentée ici :

```
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;
```

Ce code définit d'abord une occurrence de String appelée `result` qui sera utilisée pour créer la version ASCII art de l'image bitmap. Il effectue ensuite une boucle sur les pixels de l'image bitmap source. Il utilise plusieurs techniques de manipulation des couleurs (non décrites ici) pour convertir le rouge, le vert et le bleu d'un pixel en une valeur d'échelle de gris (un nombre entre 0 et 255). Le code divise ensuite cette valeur par 4 (comme indiqué) pour la convertir en une valeur dans l'échelle 0-63, qui est stockée dans la variable `index`. (L'échelle 0-63 est utilisée car la palette des caractères ASCII disponibles utilisée par cette application contient 64 valeurs.) La palette des caractères est définie en tant qu'occurrence de String dans la classe 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><+_~-;,. ";
```

Etant donné que la variable `index` définit le caractère ASCII de la palette qui correspond au pixel actuel dans l'image bitmap, ce caractère est récupéré de la `palette` String à l'aide de la méthode `charAt()`. Il est ensuite ajouté à l'occurrence de String `result` au moyen de l'opérateur d'affectation de concaténation (`+=`). En outre, à la fin de chaque ligne de pixels, un caractère de nouvelle ligne est concaténé à la fin de l'occurrence de String `result` afin que la ligne à renvoyer crée une ligne de pixels de caractères.

Chapitre 8 : Utilisation de tableaux

Les tableaux vous permettent de stocker plusieurs valeurs dans une seule structure de données. Vous pouvez utiliser des tableaux indexés simples qui stockent des valeurs à l'aide d'index d'entiers ordinaux fixes ou des tableaux associatifs complexes qui stockent des valeurs à l'aide de clés arbitraires. Les tableaux peuvent également être multidimensionnels et contenir des éléments étant eux-mêmes des tableaux. Pour finir, vous pouvez utiliser un vecteur pour les tableaux dont les éléments sont tous des occurrences du même type de données. Ce chapitre décrit comment créer et manipuler différents types de tableaux.

Principes de base des tableaux

Introduction à l'utilisation des tableaux

Vous aurez souvent besoin en programmation d'utiliser un ensemble d'éléments plutôt qu'un seul objet; par exemple, dans une application de lecteur de musique, vous pouvez avoir une liste de morceaux en attente de lecture. Vous ne souhaitez pas créer une variable séparée pour chaque morceau de cette liste. Il serait préférable de rassembler tous les objets Song et de les utiliser sous forme de groupe.

Un tableau est un élément de programmation qui agit comme conteneur pour un ensemble d'éléments (une liste de morceaux, par exemple). La plupart du temps, tous les éléments d'un tableau sont des occurrences de la même classe, mais ceci n'est pas obligatoire dans ActionScript. Les éléments individuels d'un tableau sont les *éléments* du tableau. Un tableau peut être comparé à un tiroir classeur pour des variables. Les variables peuvent être ajoutées en tant qu'éléments au tableau, comme vous placez un dossier dans le tiroir classeur. Vous pouvez utiliser le tableau comme une variable unique, comme si vous transportiez le tiroir entier à un autre endroit. Vous pouvez utiliser les variables en tant que groupe, comme si vous recherchiez des informations dans les dossiers en les parcourant l'un après l'autre. Vous pouvez y accéder individuellement, comme si vous ouvriez le tiroir et sélectionniez un seul dossier.

Par exemple, imaginez que vous créez une application de lecteur de musique dans laquelle un utilisateur peut sélectionner plusieurs morceaux et les ajouter à une liste de lecture. Dans votre code ActionScript, vous avez une méthode appelée `addSongsToPlaylist()` qui accepte un seul tableau comme paramètre. Peu importe le nombre de morceaux à ajouter à la liste (quelques-uns, un grand nombre, ou même un seul), vous devez appeler la méthode `addSongsToPlaylist()` une seule fois, en lui transmettant le tableau qui contient les objets Song. Dans la méthode `addSongsToPlaylist()`, vous pouvez utiliser une boucle pour parcourir les éléments (morceaux) du tableau un par un et les ajouter à la liste de lecture.

Le type de tableau ActionScript le plus courant est le *tableau indexé*. Dans un tableau indexé, chaque élément est stocké dans un emplacement numéroté appelé *index*. On accède à des éléments à l'aide du numéro, comme une adresse. Les tableaux indexés répondent à la plupart des besoins en programmation. La classe Array est une classe courante utilisée pour représenter un tableau indexé.

Un tableau indexé est souvent utilisé pour stocker plusieurs éléments du même type, des objets qui sont des occurrences de la même classe. La classe Array ne dispose pas de moyens pour restreindre le type d'éléments qu'elle contient. La classe Vector est un type de tableau indexé dans lequel tous les éléments d'un tableau unique sont du même type. L'utilisation d'une occurrence de Vector à la place d'une occurrence de Array peut également déboucher sur une amélioration des performances entre autres. la classe Vector est prise en charge à partir de Flash Player 10 et Adobe AIR 1.5.

Une utilisation spéciale d'un tableau indexé est un *tableau multidimensionnel*. Un tableau multidimensionnel est un tableau indexé dont les éléments sont des tableaux indexés, qui contiennent à leur tour d'autres éléments.

Le *tableau associatif* est un autre type de tableau. Il utilise une chaîne *key* au lieu d'un index numérique pour identifier des éléments individuels. Enfin, ActionScript 3.0 comprend également une classe Dictionary qui représente un *dictionnaire*. Un dictionnaire est un tableau qui vous permet d'utiliser tout type d'objet comme clé afin de distinguer les éléments entre eux.

Tâches de tableau courantes

Les activités courantes suivantes pour une utilisation des tableaux sont décrites dans ce chapitre :

- Création de tableaux indexés à l'aide des classes Array et Vector
- Ajout et suppression d'éléments de tableau
- Tri d'éléments de tableau
- Extraction de portions d'un tableau
- Utilisation de tableaux associatifs et de dictionnaires
- Utilisation de tableaux multidimensionnels
- Copie d'éléments de tableau
- Création d'une sous-classe de tableau

Concepts importants et terminologie

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Tableau : un objet qui sert de conteneur pour regrouper plusieurs objets
- Opérateur `[]` d'accès au tableau : une paire de crochets entourant un index ou une clé qui identifie de façon unique un élément du tableau. Cette syntaxe est utilisée après le nom d'une variable de tableau pour spécifier un seul élément du tableau plutôt qu'un tableau entier.
- Tableau associatif : un tableau qui utilise des clés de chaîne pour identifier des éléments individuels
- Type de base : le type de données des objets qu'une occurrence de Vector est autorisée à stocker
- Dictionnaire : un tableau dont les éléments sont constitués de paires d'objets appelées clé et valeur. La clé est utilisée à la place d'un index numérique pour identifier un seul élément.
- Elément : un élément unique dans un tableau
- Index : l'adresse numérique utilisée pour identifier un élément unique dans un tableau indexé.
- Tableau indexé : le type de tableau standard qui stocke chaque élément dans une position numérotée et utilise le numéro (index) pour identifier des éléments individuels
- Clé : la chaîne ou objet utilisé pour identifier un seul élément dans un tableau associatif ou un dictionnaire
- Tableau multidimensionnel : un tableau contenant des éléments qui sont des tableaux plutôt que des valeurs uniques
- `T` : la convention standard utilisée dans la présente documentation pour représenter le type de base d'une occurrence de Vector, quel que soit ce type de base. La convention T est utilisée pour représenter un nom de classe, comme cela est indiqué dans la description du paramètre Type. (“ T ” correspond à “ type ”, comme dans " type de données ".

- Paramètre Type : la syntaxe utilisée avec le nom de classe Vector pour spécifier le type de base de Vector (le type de données des objets qu'il stocke). La syntaxe consiste en un point (`.`), puis le nom du type de données entouré de parenthèses en chevron (`<>`). L'ensemble ressemble à ceci : `Vector<T>`. Dans la présente documentation, la classe spécifiée dans le paramètre Type est représenté de façon générique par `T`.
- Vector : un type de tableau dont les éléments sont tous des occurrences du même type de données.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester ses exemples de code. Tous les codes de ce chapitre comprennent l'appel de la fonction `trace()`. Pour tester les codes de ce chapitre :

1. Créez un document vide dans l'outil de programmation Flash
2. Sélectionnez une image-clé dans le scénario.
3. Ouvrez le panneau Actions et copiez le code dans le panneau Script.
4. Exécutez le programme en sélectionnant Contrôle >Tester l'animation.

   Les résultats de la fonction `trace()` s'affichent dans le panneau Sortie.

Ceci ainsi que d'autres techniques de test des codes sont décrits en détail à la rubrique « Test des exemples de code contenus dans un chapitre » à la page 36.

Tableaux indexés

Les tableaux indexés stockent une série d'une ou de plusieurs valeurs organisées de façon à ce que vous puissiez accéder à chaque valeur à l'aide d'une valeur d'entier non signé. Le premier index correspond toujours au nombre 0. L'index est ensuite incrémenté d'une unité pour chaque élément ajouté consécutivement au tableau. Dans ActionScript 3.0, deux classes sont utilisées comme tableaux indexés : la classe Array et la classe Vector.

Les tableaux indexés utilisent un entier 32 bits non signé pour le numéro d'index. La taille maximale d'un tableau indexé est 2^{32} - 1 ou 4 294 967 295. Si vous tentez de créer un tableau plus grand que la taille maximale, une erreur d'exécution se produit.

Pour accéder à un élément particulier d'un tableau indexé, vous pouvez utiliser l'opérateur (`[]`) d'accès au tableau pour spécifier la position de l'index de l'élément visé. Par exemple, le code suivant représente le premier élément (l'élément à l'index 0) dans un tableau indexé appelé `songTitles` :

```
songTitles[0]
```

La combinaison du nom de la variable du tableau suivi de l'index entre crochets fonctionne comme un identifiant unique. En d'autres termes, elle peut être utilisée tout comme un nom de variable. Vous pouvez affecter une valeur à un élément du tableau indexé en utilisant le nom et l'index du côté gauche d'une instruction d'affectation :

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

Dans la même veine, vous pouvez récupérer une valeur à un élément du tableau indexé en utilisant le nom et l'index du côté droit d'une instruction d'affectation :

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

Vous pouvez aussi utiliser une variable entre crochets plutôt que de fournir une valeur explicite. Elle doit contenir une valeur entière non-négative telle qu'un uint, un int positif ou une occurrence de Number d'entier positif. Cette technique est utilisée couramment pour passer en boucle sur les éléments dans un tableau indexé et effectuer une opération sur un ou tous les éléments. Le code ci-dessous décrit cette technique. Le code utilise une boucle pour accéder à chaque valeur dans un objet Array appelé `oddNumbers`. Il utilise l'instruction `trace()` pour imprimer chaque valeur sous la forme "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());
}
```

Classe Array

La classe Array est le premier type de tableau indexé. Une occurrence de Array peut comporter une valeur de n'importe quel type de données. Le même objet Array peut comporter des objets qui sont de types de données différents. Par exemple, une occurrence de Array unique peut avoir une valeur String en index 0, une occurrence de Number en index 1 et un objet XML en index 2.

Classe Vector

La classe Vector est un autre type de tableau indexé qui est disponible dans ActionScript 3.0. Une occurrence de Vector est un *tableau typé*, ce qui signifie que tous les éléments d'une occurrence de Vector ont toujours le même type de données.

Remarque : *la classe Vector est prise en charge à partir de Flash Player 10 et Adobe AIR 1.5.*

Lorsque vous déclarez une variable Vector ou que vous instanciez un objet Vector, vous spécifiez explicitement le type de données des objets que le vecteur peut contenir. Le type de données spécifié est connu sous le nom de *type de base* du vecteur. Lors de l'exécution et de la compilation (en mode strict), tout code qui fixe la valeur d'un élément Vector ou récupère une valeur d'un élément Vector est contrôlé. Si le type de données de l'objet que l'on tente d'ajouter ou de récupérer ne correspond pas au type de base du vecteur, une erreur se produit.

Outre la restriction concernant le type de données, la classe Vector possède d'autres restrictions qui la distinguent de la classe Array :

- Un vecteur est un tableau dense. Un objet Array peut comporter des valeurs dans les index 0 et 7 même si elle n'en a pas dans les positions 1 à 6. Cependant, un vecteur doit comporter une valeur (ou `null`) dans chaque index.
- Un vecteur peut facultativement avoir une longueur fixe. Ceci signifie que le nombre d'éléments du vecteur est immuable.
- L'accès aux éléments d'un vecteur est défini par ses limites. Vous ne pouvez jamais lire une valeur d'un index supérieur à celui de l'élément final (`longueur` - 1). Vous ne pouvez jamais définir une valeur avec un index supérieur à l'index final actuel. En d'autres termes, vous pouvez définir une valeur uniquement à l'index existant où à une `[longueur]` d'index.

Grâce à ses restrictions, un vecteur présente deux avantages principaux par rapport à une occurrence de Array dont les éléments sont tous des occurrences d'une seule classe :

- Performance : l'accès à l'élément de tableau et son itération sont beaucoup plus rapides lorsque vous utilisez une occurrence de Vector que lorsque vous utilisez une occurrence d'Array.

- Sécurité des types : en mode strict, le compilateur peut identifier les erreurs de type de données. Parmi ces erreurs, il y a l'affectation d'une valeur du type de données incorrect à un vecteur ou l'attente d'un type de données incompatible lors de la lecture d'une valeur à partir du vecteur. A l'exécution, les types de données sont également contrôlés lors de l'ajout de données à un objet Vector ou la lecture de données qui en provient. Notez cependant que lorsque vous utilisez la méthode `push()` ou `unshift()` pour ajouter des valeurs à un vecteur, les types de données des arguments ne sont pas vérifiés au moment de la compilation. Lorsque vous utilisez ces méthodes, les valeurs sont toujours contrôlées à l'exécution.

A part les contraintes et les avantages supplémentaires, la classe Vector est très proche de la classe Array. Les propriétés et les méthodes d'un objet Vector sont similaires, voire dans certains cas identiques, aux propriétés et aux méthodes d'un objet Array. Dans tous les cas où vous utilisez un objet Array dont tous les éléments possèdent le même type de données, il est préférable d'utiliser une occurrence de l'objet Vector.

Création de tableaux

Vous pouvez utiliser plusieurs techniques pour créer une occurrence de Array ou une occurrence de Vector. Cependant, les techniques de création de chaque type de tableau sont quelque peu différentes.

Création d'une occurrence de Array

Vous créez un objet Array par l'appel au constructeur `Array(` ou par l'utilisation d'une syntaxe de littéral de tableau.

Vous pouvez utiliser la fonction de constructeur `Array()` de trois façons différentes. Premièrement, si vous appelez le constructeur sans arguments, vous obtenez un tableau vide. Vous pouvez utiliser la propriété `length` de la classe Array pour vérifier que le tableau ne contient aucun élément. Par exemple, le code suivant appelle le constructeur `Array()` sans arguments :

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

Deuxièmement, si vous utilisez un nombre comme unique paramètre pour le constructeur `Array()`, un tableau de cette longueur est créé, avec chaque valeur d'élément définie sur `undefined`. L'argument doit être un entier non signé compris entre les valeurs 0 et 4 294 967 295. Par exemple, le code suivant appelle le constructeur `Array()` avec un seul argument numérique :

```
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
```

Troisièmement, si vous appelez le constructeur et transmettez une liste d'éléments comme paramètres, un tableau est créé avec des éléments correspondant à chacun des paramètres. Le code suivant transmet trois arguments au constructeur `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
```

Vous pouvez aussi créer des tableaux avec des littéraux de tableau. Un littéral de tableau peut être affecté directement à une variable de tableau, comme illustré dans l'exemple suivant :

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

Création d'une occurrence de Vector

Vous créez une occurrence de Vector par l'appel du constructeur `Vector.<T>()`. Vous pouvez aussi créer un vecteur par l'appel à la fonction globale `Vector.<T>()`. Cette fonction convertit un objet spécifié en une occurrence de Vector. ActionScript n'a pas d'équivalent Vector à la syntaxe de littéral de tableau.

Toutes les fois que vous déclarez une variable Vector (ou de la même façon, un paramètre de la méthode Vector ou un type de renvoi de la méthode Vector), vous spécifiez le type de base de la variable Vector. Vous spécifiez également le type de base lorsque vous créez une occurrence de Vector par l'appel au constructeur `Vector.<T>()`. Autrement dit, toutes les fois que vous utilisez le terme `Vector` dans ActionScript, il est accompagné d'un type de base.

Vous spécifiez le type de base du vecteur à l'aide de la syntaxe de paramètres de type. Le paramètre de type suit immédiatement le mot `Vector` dans le code. Il est formé d'un point (`.`), puis du nom de classe de base entouré de parenthèses en chevron (`<>`), comme l'indique cet exemple :

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

Dans la première ligne de cet exemple, la variable `v` est déclarée comme l'occurrence d'un objet Vector.`Occurrence <String>`. En d'autres termes, il représente un tableau indexé qui ne peut comporter que des occurrences String. La deuxième ligne appelle le constructeur `Vector()` pour créer une occurrence du même type Vector, c'est-à-dire un vecteur dont les éléments sont tous des objets String. Il affecte cet objet à `v`.

Utilisation du constructeur Vector.<T>()

Si vous utilisez le constructeur `Vector.<T>()` sans arguments, il crée une occurrence de Vector vide. Vous pouvez contrôler qu'un vecteur est vide en vérifiant sa propriété `length`. Par exemple, le code ci-dessus appelle le constructeur `Vector.<T>()` sans arguments :

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

Si vous savez d'avance de combien d'éléments un vecteur a besoin initialement, vous pouvez prédéfinir ce nombre dans le vecteur. Pour créer un vecteur avec un certain nombre d'éléments, transmettez le nombre d'éléments comme premier paramètre (le paramètre `length`). Comme les éléments du vecteur ne peuvent pas être vides, ils sont remplis d'occurrences du type de base. Si ce type est un type de référence qui autorise les valeurs `null`, alors les éléments contiennent tous `null`. Autrement, ils contiennent tous la valeur par défaut pour la classe. Par exemple, une variable uint ne peut pas être `null`. Par conséquent, dans le code ci-dessous, le vecteur appelé `ages` est créé avec sept éléments, chacun contenant la valeur 0.

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

Enfin, à l'aide du constructeur `Vector.<T>()`, vous pouvez également créer un vecteur de longueur fixe en transmettant `true` comme deuxième paramètre (le paramètre `fixed`). Dans ce cas, le vecteur est créé avec le nombre spécifié d'éléments et celui-ci ne peut pas être modifié. Notez cependant que vous pouvez quand même changer les valeurs des éléments d'un vecteur de longueur fixe.

Contrairement à la classe Array, vous ne pouvez pas transmettre une liste de valeurs au constructeur `Vector.<T>()` pour spécifier les valeurs initiales du vecteur.

Utilisation de la fonction globale Vector. <T>()

Outre le constructeur `Vector.<T>()`, vous pouvez également utiliser la fonction globale `Vector. <T>()` pour créer un objet Vector. La fonction globale `Vector.<T>()` est une fonction de conversion. Lorsque vous appelez la fonction globale `Vector.<T>()`, vous spécifiez le type de base du vecteur que la méthode renvoie. Vous transmettez un tableau indexé unique (occurrence de Array ou Vector) comme argument. La méthode renvoie alors un vecteur avec le type de base spécifié, contenant les valeurs dans l'argument du tableau source. Le code ci-dessous montre la syntaxe nécessaire pour appeler la fonction globale `Vector.<T>()`.

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

La fonction globale `Vector.<T>()` exécute une conversion de type de base sur deux niveaux. D'abord, lorsqu'une occurrence de Array est transmise à la fonction, une occurrence de Vector est renvoyée. Ensuite, que le tableau source soit une occurrence de Array ou Vector, la fonction tente de convertir les éléments du tableau source en valeurs du type de base. La conversion utilise des règles standard de conversion des types de données ActionScript. Par exemple, le code suivant convertit les valeurs String du tableau source en nombres entiers dans le vecteur résultant. La partie décimale de la première (`"1.5"`) est tronquée et la troisième valeur non-numérique (`"Waffles"`) est convertie en 0 dans le résultat :

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

S'il n'est pas possible de convertir un élément source quelconque, une erreur se produit.

Lorsque le code appelle la fonction globale `Vector.<T>()`, si un élément du tableau source est une occurrence de sous-classe du type de base spécifié, l'élément est ajouté au vecteur résultant (aucune erreur ne se produit). L'utilisation de la fonction globale `Vector.<T>()` est en fait le seul moyen de convertir un vecteur avec un type de base `T` en un vecteur avec un type de base qui est une superclasse de `T`.

Insertion d'éléments de tableau

L'opérateur (`[]`) d'accès au tableau constitue le moyen le plus élémentaire d'ajouter un élément à un tableau indexé. Pour définir la valeur d'un élément de tableau indexé, utilisez le nom d'objet Array ou Vector et le numéro d'index du côté gauche d'une instruction d'affectation.

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

Si un élément ne se trouve pas déjà à cette position d'index du tableau ou du vecteur, l'index est créé et la valeur y est stockée. Si une valeur existe à cet index, la nouvelle valeur remplace l'ancienne.

Un objet Array vous permet de créer un élément pour tout index. Cependant, avec un objet Vector, vous pouvez affecter uniquement une valeur à un index existant ou à l'index disponible suivant. Celui-ci correspond à la propriété `length` de l'objet Vector. Utilisez du code comme celui qui est présenté ci-dessous pour ajouter un nouvel élément à un objet Vector de la façon la plus sûre :

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

Trois méthodes de la classe Array —`push()`, `unshift()`, et `splice()`—vous permettent d'insérer des éléments dans un tableau indexé. La méthode `push()` ajoute un ou plusieurs éléments à la fin d'un tableau. Ainsi, le dernier élément inséré dans le tableau à l'aide de la méthode `push()` aura le numéro d'index le plus élevé. La méthode `unshift()` insère un ou plusieurs éléments au début d'un tableau, qui est toujours au numéro d'index 0. La méthode `splice()` insère des éléments au niveau d'un index spécifié dans le tableau.

L'exemple suivant illustre les trois méthodes. Un tableau appelé `planets` est créé pour trier les noms des planètes par ordre de proximité par rapport au soleil. La méthode `push()` est tout d'abord appelée pour ajouter l'élément initial, `Mars`. Deuxièmement, la méthode `unshift()` est appelée pour insérer l'élément `Mercury`. Pour finir, la méthode `splice()` est appelée pour insérer les éléments `Venus` et `Earth` après `Mercury`, mais avant `Mars`. Le premier élément envoyé à `splice()`, l'entier 1, indique à l'insertion de débuter à l'index 1. Le deuxième argument envoyé à `splice()`, l'entier 0, indique qu'aucun élément ne doit être supprimé. Pour finir, les troisième et quatrième arguments envoyés à `splice()`, `Venus` et `Earth`, sont les éléments à insérer.

```
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
```

Les méthodes `push()` et `unshift()` renvoient toutes les deux un entier non signé qui représente la longueur du tableau modifié. La méthode `splice()` renvoie un tableau vide lorsqu'elle est utilisée pour insérer des éléments, ce qui semble étrange mais compréhensible en raison de sa versatilité. Vous pouvez utiliser la méthode `splice()` non seulement pour insérer des éléments dans un tableau, mais également pour en supprimer. Lorsque vous l'utilisez pour supprimer des éléments, la méthode `splice()` renvoie un tableau contenant les éléments supprimés.

Remarque : *si une propriété `fixed` de l'objet Vector est définie sur `true`, le nombre total d'éléments du vecteur reste immuable. Si vous tentez d'ajouter un nouvel élément à un vecteur de longueur fixe à l'aide des techniques décrites ici, une erreur se produit.*

Récupération des valeurs et suppression des éléments du tableau

Utilisez l'opérateur (`[]`) d'accès au tableau pour récupérer la valeur d'un élément de la façon la plus simple. Pour récupérer la valeur d'un élément de tableau indexé, utilisez le nom d'objet Array ou Vector et le numéro d'index du côté droit d'une instruction d'affectation.

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

Il est possible d'essayer de récupérer une valeur à partir d'un tableau ou d'un vecteur à l'aide d'un index où aucun élément n'existe. Dans ce cas, un objet Array renvoie la valeur non définie et un vecteur renvoie une exception RangeError.

Trois méthodes des classes Array et Vector s—`pop()`, `shift()` et `splice()`—vous permettent de supprimer des éléments. La méthode `pop()` supprime un élément de la fin du tableau. En d'autres termes, elle supprime l'élément au niveau du numéro d'index le plus élevé. La méthode `shift()` supprime un élément du début du tableau, ce qui signifie qu'elle supprime toujours l'élément au numéro d'index 0. La méthode `splice()`, qui peut également être utilisée pour insérer des éléments, supprime un nombre arbitraire d'éléments en commençant au numéro d'index indiqué par le premier argument envoyé à la méthode.

L'exemple suivant utilise les trois méthodes pour supprimer des éléments d'une occurrence de Array. Un tableau nommé `oceans` est créé pour stocker les noms des océans. Certains noms dans le tableau sont des noms de lacs plutôt que des noms d'océans. Ils doivent donc être supprimés.

Premièrement, la méthode `splice()` est utilisée pour supprimer les éléments `Aral` et `Superior`, et insérer les éléments `Atlantic` et `Indian`. Le premier argument envoyé à `splice()`, l'entier 2, indique que l'opération doit commencer par le troisième élément dans la liste, qui est à l'index 2. Le deuxième argument, 2, indique que deux éléments doivent être supprimés. Les arguments restants, `Atlantic` et `Indian`, sont des valeurs à insérer à l'index 2.

Deuxièmement, la méthode `pop()` est utilisée pour supprimer le dernier élément dans le tableau, `Huron`. Et troisièmement, la méthode `shift()` est utilisée pour supprimer le premier élément dans le tableau, `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
```

Les méthodes `pop()` et `shift()` renvoient toutes les deux l'élément qui a été supprimé. Pour une occurrence de Array, le type de données de la valeur renvoyée est Object car les tableaux peuvent contenir des valeurs de n'importe quel type de données. Pour une occurrence de Vector, le type de données de la valeur renvoyée est le type de base du vecteur. La méthode `splice()` renvoie un tableau ou un vecteur contenant les valeurs supprimées. Vous pouvez modifier l'exemple du tableau `oceans` de façon à ce que l'appel à `splice()` affecte le tableau renvoyé à une nouvelle variable de tableau, comme illustré dans l'exemple suivant :

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

Il se peut que vous rencontriez un code qui utilise l'opérateur `delete` sur un élément de l'objet Array. L'opérateur `delete` définit la valeur d'un élément de tableau sur `undefined`, mais il ne supprime pas l'élément du tableau. Par exemple, le code suivant utilise l'opérateur `delete` sur le troisième élément dans le tableau `oceans`, mais la longueur du tableau demeure à 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
```

Vous pouvez tronquer un tableau à l'aide d'une propriété `length` de tableau ou de vecteur. Si vous définissez la propriété `length` d'un tableau indexé sur une longueur qui est moindre que la longueur courante du tableau, celui-ci est tronqué : tous les éléments stockés à des numéros d'index supérieurs à la nouvelle valeur `length`, diminuée de 1, sont supprimés. Par exemple, si le tableau `oceans` était trié de telle façon que toutes les entrées valides se trouvaient au début du tableau, vous pourriez utiliser la propriété `length` pour supprimer les entrées de fin de tableau, comme l'indique le code ci-dessous :

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

Remarque : *si une propriété `fixed` de l'objet Vector est définie sur `true`, le nombre total d'éléments du vecteur reste immuable. Si vous essayez de supprimer un élément d'un vecteur de longueur fixe ou de tronquer celui-ci à l'aide des techniques décrites ici, une erreur se produit.*

Tri d'un tableau

Il existe trois méthodes—`reverse()`, `sort()` et `sortOn()`—qui vous permettent de modifier l'ordre d'un tableau indexé, soit en triant, soit en inversant l'ordre. Toutes ces méthodes modifient le tableau existant. Le tableau ci-dessous résume ces méthodes et leurs comportements pour les objets Array et Vector :

Méthode	Comportement d'Array	Comportement de Vector
`reverse()`	Modifie l'ordre des éléments de telle sorte que le dernier élément devient le premier élément, le pénultième le deuxième, etc.	Identique au comportement d'Array
`sort()`	Vous permet de trier les éléments du tableau de diverses façons prédéfinies, comme l'ordre alphabétique ou numérique. Vous pouvez également spécifier un algorithme de tri personnalisé.	Trie les éléments suivant l'algorithme de tri personnalisé que vous spécifiez
`sortOn()`	Vous permet de trier des objets qui ont une ou plusieurs propriétés en commun en spécifiant la ou les propriétés à utiliser comme critères de tri.	Non disponible dans la classe Vector

Méthode reverse()

La méthode `reverse()` ne prend aucun paramètre et ne renvoie aucune valeur mais vous permet de faire basculer l'ordre de votre tableau de son état actuel à l'ordre inverse. L'exemple suivant inverse l'ordre des océans répertoriés dans le tableau `oceans` :

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

Tri de base avec la méthode sort() (classe Array uniquement)

Pour une occurrence de Array, la méthode `sort()` réorganise les éléments dans un tableau à l'aide de l'*ordre de tri par défaut*. L'ordre de tri par défaut possède les caractéristiques suivantes :

- Le tri est sensible à la casse, ce qui signifie que les majuscules précédent les minuscules. Par exemple, la lettre D précède la lettre b.
- Le tri est croissant, ce qui signifie que les codes de caractère bas (A, par exemple) précédent les codes de caractère élevés (B, par exemple).
- Le tri place les valeurs identiques les unes à côté des autres mais sans ordre particulier.
- Le tri est basé sur des chaînes, ce qui signifie que les éléments sont convertis en chaînes avant d'être comparés (par exemple, 10 précède 3 car la chaîne `"1"` a un code de caractère inférieur à celui de la chaîne `"3"`).

Vous pouvez trier votre tableau en ignorant la casse ou par ordre décroissant. Vous pouvez également trier les nombres de votre tableau par ordre numérique plutôt que par ordre alphabétique. La méthode `sort()` de la classe Array possède un paramètre `options` qui vous permet de modifier chaque caractéristique de l'ordre de tri par défaut. Les options sont définies par un ensemble de constantes statiques dans la classe Array, comme indiqué dans la liste suivante :

- `Array.CASEINSENSITIVE` : cette option permet d'ignorer la casse lors du tri. Par exemple, la lettre minuscule b précède la lettre majuscule D.
- `Array.DESCENDING` : cette option inverse le tri croissant par défaut. Par exemple, la lettre B précède la lettre A.
- `Array.UNIQUESORT` : cette option permet d'arrêter le tri si deux valeurs identiques sont repérées.
- `Array.NUMERIC` : cette option permet d'effectuer un tri numérique, de façon à ce que 3 précède 10.

L'exemple suivant met en évidence certaines de ces options. Un tableau appelé `poets` est créé. Il est trié à l'aide de plusieurs options différentes.

```
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
```

Tri personnalisé avec la méthode sort() (classes Array et Vector)

En plus du tri de base qui est disponible pour un objet Array, vous pouvez également établir une règle de tri personnalisée. Cette technique est la seule forme de méthode `sort()` disponible pour la classe Vector. Pour définir un tri personnalisé, vous rédigez une fonction de tri personnalisée et la transmettez comme argument à la méthode `sort()`.

Par exemple, si vous avez une liste de noms dans laquelle chaque élément de liste contient le nom entier d'une personne mais que vous souhaitez trier la liste par nom de famille, vous devez utiliser une fonction de tri personnalisé pour analyser chaque élément et utiliser le nom de famille dans la fonction de tri. Le code suivant indique comment procéder avec une fonction personnalisée utilisée comme paramètre pour la méthode `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
```

La fonction de tri personnalisé `orderLastName()` utilise une expression régulière pour extraire le nom de famille de chaque élément à utiliser pour l'opération de comparaison. L'identifiant de fonction `orderLastName` est utilisé comme paramètre lors de l'appel à la méthode `sort()` sur le tableau `names`. La fonction de tri accepte deux paramètres, `a` et `b`, car elle fonctionne sur deux éléments de tableau à la fois. La valeur renvoyée de la fonction de tri indique la manière dont les éléments doivent être triés :

- Une valeur renvoyée de -1 indique que le premier paramètre, `a`, précède le second paramètre, `b`.
- Une valeur renvoyée de 1 indique que le second paramètre, `ab`, précède le premier, `a`.

- Une valeur renvoyée de 0 indique que les éléments ont une précédence de tri équivalente.

Méthode sortOn() (classe Array uniquement)

La méthode `sortOn()` est conçue pour des objets Array avec des éléments contenant des objets. Ces objets doivent avoir au moins une propriété en commun pouvant être utilisée comme clé de tri. L'utilisation de la méthode `sortOn()` pour des tableaux d'autres types provoque des résultats inattendus.

Remarque : *la classe Vector ne contient pas de méthode `sortOn()`. Cette méthode n'est disponible que pour les objets Array.*

L'exemple suivant modifie le tableau `poets` de façon à ce que chaque élément soit un objet plutôt qu'une chaîne. Chaque objet contient à la fois le nom de famille du poète et sa date de naissance.

```
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"});
```

Vous pouvez utiliser la méthode `sortOn()` pour trier le tableau par la propriété `born`. La méthode `sortOn()` définit deux paramètres, `fieldName` et `options`. L'argument `fieldName` doit être spécifié en tant que chaîne. Dans l'exemple suivant, la méthode `sortOn()` est appelée avec deux arguments, `"born"` et `Array.NUMERIC`. L'argument `Array.NUMERIC` est utilisé pour vérifier que le tri est effectué par ordre numérique plutôt que par ordre alphabétique. Ceci est utile même lorsque tous les nombres ont le même nombre de chiffres car vous êtes certain que le tri se fera comme prévu si un nombre comportant un nombre inférieur ou supérieur de chiffres est ensuite ajouté au tableau.

```
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
*/
```

Tri sans modification du tableau d'origine (classe Array uniquement)

Généralement, les méthodes `sort()` et `sortOn()` modifient un tableau. Si vous souhaitez trier un tableau sans modifier le tableau existant, transmettez la constante `Array.RETURNINDEXEDARRAY` avec le paramètre `options`. Cette option indique aux méthodes de renvoyer un nouveau tableau qui reflète le tri et laisse le tableau d'origine inchangé. Le tableau renvoyé par les méthodes est un tableau simple de numéros d'index qui reflète le nouvel ordre de tri et ne contient aucun élément du tableau d'origine. Par exemple, pour trier le tableau `poets` par année de naissance sans le modifier, incluez la constante `Array.RETURNINDEXEDARRAY` dans l'argument transmis pour le paramètre `options`.

L'exemple suivant stocke les informations d'index renvoyées dans un tableau nommé `indices` et utilise le tableau `indices` avec le tableau `poets` inchangé pour trier les poètes dans l'ordre de leur année de naissance :

```
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
*/
```

Interrogation d'un tableau

Les quatre méthodes restantes des classes Array et Vector—`concat()`, `join()`, `slice()`, `toString()`—interrogent toutes le tableau mais ne le modifient pas. Les méthodes `concat()` et `slice()` renvoient toutes les deux de nouveaux tableaux, alors que les méthodes `join()` et `toString()` renvoient des chaînes. La méthode `concat()` prend un nouveau tableau ou une liste d'éléments comme arguments et le/la combine avec le tableau existant pour créer un tableau. La méthode `slice()` possède deux paramètres nommés `startIndex` et `endIndex`, et renvoie un nouveau tableau contenant une copie des éléments découpés du tableau existant. La découpe commence avec l'élément à `startIndex` et se termine avec l'élément juste avant `endIndex`. Il convient d'insister : l'élément à `endIndex` n'est pas compris dans la valeur renvoyée.

L'exemple suivant utilise `concat()` et `slice()` pour créer des tableaux à l'aide d'éléments d'autres tableaux :

```
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
```

Vous pouvez utiliser les méthodes `join()` et `toString()` pour interroger le tableau et renvoyer son contenu sous la forme d'une chaîne. Si aucun paramètre n'est utilisé pour la méthode `join()`, les deux méthodes se comportent de façon identique : elles renvoient une chaîne contenant une liste de tous les éléments du tableau séparée par des virgules. La méthode `join()`, contrairement à la méthode `toString()`, accepte un paramètre nommé `delimiter`, qui permet de choisir le symbole à utiliser comme séparateur entre chaque élément de la chaîne renvoyée.

L'exemple suivant crée un tableau nommé `rivers` et appelle à la fois `join()` et `toString()` pour renvoyer les valeurs dans le tableau sous la forme d'une chaîne. La méthode `toString()` est utilisée pour renvoyer des valeurs séparées par une virgule (`riverCSV`), alors que la méthode `join()` est utilisée pour renvoyer des valeurs séparées par le caractère `+`.

```
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
```

Il existe un problème avec la méthode `join()` ; toutes les occurrences de tableau et de vecteur imbriquées sont toujours renvoyées avec des valeurs séparées par des virgules, quel que soit le séparateur que vous spécifiez pour les éléments de tableau principaux, comme illustré dans l'exemple suivant :

```
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
```

Tableaux associatifs

Un tableau associatif, parfois appelé *hachage* ou *mappage*, utilise des *clés* plutôt qu'un index numérique pour organiser des valeurs stockées. Chaque clé dans un tableau associatif est une chaîne unique qui est utilisée pour accéder à une valeur stockée. Un tableau associatif est une occurrence de la classe Object, ce qui signifie que chaque clé correspond à un nom de propriété. Les tableaux associatifs sont des collections non triées de paires de clés et de valeurs. Votre code ne doit pas s'attendre à ce que les clés d'un tel tableau se présentent dans un ordre précis.

ActionScript 3.0 contient aussi un type avancé de tableau associatif appelé *dictionnaire*. Les dictionnaires, qui sont des occurrences de la classe Dictionary dans le package flash.utils, utilisent des clés de tout type de données. En d'autres termes, les clés de dictionnaire ne sont pas limitées à des valeurs de type String.

Tableaux associatifs avec clés de chaîne

Deux méthodes permettent de créer des tableaux associatifs dans ActionScript 3.0. La première consiste à utiliser une occurrence de Object. Celle-ci vous permet d'initialiser votre tableau avec un littéral d'objet. Une occurrence de la classe Object, également appelée *objet générique*, présente le même fonctionnement qu'un tableau associatif. Chaque nom de propriété de l'objet générique devient la clé qui permet d'accéder à une valeur stockée.

L'exemple suivant crée un tableau associatif appelé `monitorInfo`, à l'aide d'un littéral d'objet pour initialiser le tableau avec deux paires de clés et de valeurs :

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

Si vous n'avez pas besoin d'initialiser le tableau lors de la déclaration, vous pouvez utiliser le constructeur Object pour créer le tableau, comme suit :

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

Une fois que le tableau est créé à l'aide d'un littéral d'objet ou du constructeur de la classe Object, vous pouvez lui ajouter de nouvelles valeurs à l'aide de l'opérateur (`[]`) d'accès au tableau ou de l'opérateur point (`.`). L'exemple suivant ajoute deux nouvelles valeurs à `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
```

La clé appelée `aspect ratio` contient un caractère d'espace. Cela est possible dans le cas de l'opérateur (`[]`) d'accès au tableau, mais avec l'opérateur point, une erreur se produit. L'utilisation d'espace dans le nom de vos clés n'est donc pas conseillée.

La seconde méthode pour créer un tableau associatif consiste à utiliser le constructeur Array (ou le constructeur d'une classe dynamique), puis à utiliser l'opérateur (`[]`) d'accès au tableau ou l'opérateur point (`.`) pour ajouter les paires de clés et de valeurs au tableau. Si vous déclarez votre tableau associatif comme étant de type Array, vous ne pouvez pas utiliser de littéral d'objet pour l'initialiser. Ce code crée un tableau associatif appelé `monitorInfo` à l'aide du constructeur Array, et ajoute les clés appelées `type` et `resolution`,ainsi que leurs valeurs :

```
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
```

L'utilisation du constructeur Array pour créer un tableau associatif ne présente aucun avantage. Vous ne pouvez pas utiliser la propriété `Array.length` ou une méthode de la classe Array avec des tableaux associatifs, même si vous utilisez le constructeur Array ou le type de données Array. Il est préférable d'utiliser le constructeur Array pour créer des tableaux indexés.

Tableaux associatifs avec clés d'objet (Dictionnaires)

Vous pouvez utiliser la classe Dictionary pour créer un tableau associatif qui utilise des objets pour les clés au lieu de chaînes. Ces tableaux sont parfois appelés dictionnaires, hachages ou mappages. Par exemple, supposez que vous avez une application qui détermine l'emplacement d'un objet Sprite selon son association avec un conteneur spécifique. Vous pouvez utiliser un objet Dictionary pour mapper chaque objet Sprite à un conteneur.

Le code suivant crée trois occurrences de la classe Sprite qui servent de clés pour l'objet Dictionary. Une valeur de `GroupA` ou `GroupB` est affectée à chaque clé. Les valeurs peuvent être de n'importe quel type de données, mais dans cet exemple, `GroupA` et `GroupB` sont des occurrences de la classe Object. Vous pouvez ensuite accéder à la valeur associée à chaque clé avec l'opérateur d'accès au tableau (`[]`), comme illustré dans le code suivant :

```
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");
}
```

Itération avec des clés d'objet

Vous pouvez parcourir en boucle le contenu d'un objet Dictionary à l'aide d'une boucle `for..in` ou d'une boucle `for each..in`. Une boucle `for..in` vous permet d'effectuer une itération en fonction des clés, tandis qu'une boucle `for each..in` vous permet d'effectuer une itération en fonction des valeurs associées à chaque clé.

Utilisez la boucle `for..in` pour accéder directement aux clés d'objet d'un objet Dictionary. Vous pouvez également accéder aux valeurs de l'objet Dictionary avec l'opérateur d'accès au tableau (`[]`). Le code suivant utilise l'exemple précédent du dictionnaire `groupMap` pour indiquer comment parcourir en boucle un objet Dictionary avec la boucle `for..in`:

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

Utilisez la boucle `for each..in` pour accéder directement aux valeurs d'un objet Dictionary. Le code suivant utilise également le dictionnaire `groupMap` pour indiquer comment parcourir en boucle un objet Dictionary avec la boucle `for each..in`:

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

Clés d'objet et gestion de la mémoire

Adobe® Flash® Player et Adobe® AIR™ utilisent un système de nettoyage permettant de récupérer la mémoire qui n'est plus utilisée. Lorsque aucune référence ne pointe vers un objet, ce dernier peut être nettoyé et la mémoire récupérée au prochain nettoyage. Par exemple, le code suivant crée un objet et lui affecte une référence à la variable `myObject` :

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

Tant que des références à l'objet existent, le système de nettoyage ne récupère pas la mémoire que l'objet occupe. Si la valeur de `myObject` est modifiée et qu'elle pointe vers un autre objet ou qu'elle est définie sur `null`, la mémoire occupée par l'objet d'origine peut être nettoyée. Néanmoins, aucune autre référence à l'objet d'origine ne doit exister.

Si vous utilisez `myObject` comme clé dans un objet Dictionary, vous créez une autre référence à l'objet d'origine. Par exemple, le code suivant crée deux références à un objet, la variable `myObject` et la clé dans l'objet `myMap` :

```
import flash.utils.Dictionary;

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

Pour que l'objet référencé par `myObject` puisse être nettoyé, vous devez supprimer toutes ses références. Dans ce cas, vous devez modifier la valeur de `myObject` et supprimer la clé `myObject` de `myMap`, comme indiqué dans le code suivant :

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

Vous pouvez également utiliser le paramètre `useWeakReference` du constructeur Dictionary pour que toutes les clés de dictionnaire deviennent des *références faibles*. Le système de nettoyage ignore les références faibles. Par conséquent, un objet n'ayant que des références faibles peut être nettoyé. Par exemple, dans le code suivant, vous n'avez pas besoin de supprimer la clé `myObject` de `myMap` pour que l'objet puisse être nettoyé :

```
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.
```

Tableaux multidimensionnels

Les tableaux multidimensionnels contiennent d'autres tableaux comme éléments. Prenons par exemple une liste de tâches stockée sous forme de tableau indexé de chaînes :

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

Pour stocker une liste de tâches distincte pour chaque jour de la semaine, vous pouvez créer un tableau multidimensionnel avec un élément pour chaque jour. Chaque élément contient à son tour un tableau indexé (semblable au tableau `tasks`) qui stocke la liste des tâches. Vous pouvez utiliser n'importe quelle combinaison de tableaux indexés ou associatifs dans des tableaux multidimensionnels. Les exemples des sections suivantes utilisent soit deux tableaux indexés soit un tableau associatif de tableaux indexés. Vous pouvez essayer les autres combinaisons pour vous exercer.

Deux tableaux indexés

Lorsque vous utilisez deux tableaux indexés, vous pouvez visualiser le résultat sous forme de tableau ou de feuille de calcul. Les éléments du premier tableau représentent les lignes alors que les éléments du second tableau représentent les colonnes.

Par exemple, le tableau multidimensionnel suivant utilise deux tableaux indexés pour suivre des listes de tâches pour chaque jour de la semaine. Le premier tableau, `masterTaskList`, est créé à l'aide du constructeur de classe Array. Chaque élément du tableau représente un jour de la semaine, avec l'index 0 représentant lundi et l'index 6 dimanche. Ces éléments peuvent être considérés comme les lignes du tableau. Vous pouvez créer la liste de tâches de chaque jour en affectant un littéral de tableau à chacun des sept éléments que vous créez dans le tableau `masterTaskList`. Les littéraux de tableau représentent les colonnes du tableau.

```
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"];
```

Vous pouvez accéder à des éléments particuliers sur toute liste des tâches à l'aide de l'opérateur d'accès au tableau (`[]`). Le premier groupe de crochets représente le jour de la semaine et le second la liste de tâches pour ce jour. Par exemple, pour récupérer la seconde tâche de la liste du mercredi, utilisez d'abord l'index 2 pour mercredi puis utilisez l'index 1 pour la seconde tâche dans la liste.

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

Pour récupérer la première tâche de la liste du dimanche, utilisez l'index 6 pour dimanche et l'index 0 pour la première tâche sur la liste.

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

Tableau associatif avec un tableau indexé

Pour faciliter l'accès aux tableaux, vous pouvez utiliser un tableau associatif pour les jours de la semaine et un tableau indexé pour les listes de tâche. Les tableaux associatifs vous permettent d'utiliser une syntaxe à point lorsque vous vous référez à un jour particulier de la semaine, mais nécessitent un traitement d'exécution supplémentaire pour accéder à chaque élément du tableau associatif. L'exemple suivant utilise un tableau associatif comme base d'une liste de tâches, avec une paire de clés et de valeurs pour chaque jour de la semaine :

```
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"];
```

La syntaxe à point rend le code plus lisible car elle évite d'utiliser plusieurs groupes de crochets.

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

Vous pouvez parcourir en boucle la liste des tâches en utilisant une boucle `for..in`, mais vous devez utiliser l'opérateur d'accès au tableau (`[]`), en lieu et place de la syntaxe à point, pour accéder à la valeur associée à chaque clé. Etant donné que `masterTaskList` est un tableau associatif, les éléments ne sont pas nécessairement récupérés dans l'ordre que vous attendez, comme l'indique l'exemple suivant :

```
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
*/
```

Clonage de tableaux

La classe Array ne possède aucune méthode intégrée pour effectuer des copies de tableaux. Vous pouvez créer une *copie simple* d'un tableau en appelant la méthode `concat()` ou `slice()` sans arguments. Dans une copie simple, si le tableau d'origine a des éléments qui sont des objets, seules les références aux objets sont copiées (et non les objets). La copie pointe vers les mêmes objets que l'original. Tout changement effectué sur les objets apparaît dans les deux tableaux.

Dans une *copie en profondeur*, les objets se trouvant dans le tableau d'origine sont copiés également de façon à ce que le nouveau tableau ne pointe pas vers les mêmes objets que le tableau d'origine. La copie en profondeur exige plus d'une ligne de code, ce qui nécessite généralement la création d'une fonction. Une telle fonction peut être créée comme fonction d'utilitaire globale ou comme méthode d'une sous-classe Array.

L'exemple suivant définit une fonction appelée `clone()` qui effectue une copie en profondeur. L'algorithme est issu d'une technique de programmation Java courante. La fonction crée une copie en profondeur en sérialisant le tableau en une occurrence de la classe ByteArray puis en relisant le tableau dans un nouveau tableau. Cette fonction accepte un objet de façon à ce qu'il puisse être utilisé à la fois avec des tableaux indexés et des tableaux associatifs, comme indiqué dans le code suivant :

```
import flash.utils.ByteArray;

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

Rubriques avancées

Extension de la classe Array

La classe Array est l'une des classes de base non finale, c'est-à-dire que vous pouvez créer votre sous-classe d'Array. Cette section décrit comment créer une sous-classe d'Array et décrit les problèmes pouvant se poser pendant le processus.

Comme mentionné précédemment, les tableaux dans ActionScript ne sont pas typés, mais vous pouvez créer une sous-classe d'Array qui accepte des éléments d'un seul type de données. L'exemple fourni dans les sections suivantes définit une sous-classe Array appelée TypedArray qui limite ses éléments à des valeurs du type de données indiqué dans le premier paramètre. La classe TypedArray est présentée comme un exemple de la façon dont la classe Array est étendue et risque de ne pas être adaptée à des fins de production pour différentes raisons. Premièrement, la vérification du type a lieu lors de l'exécution plutôt que de la compilation. Deuxièmement, lorsqu'une méthode TypedArray rencontre une incompatibilité, elle est ignorée et aucune exception n'est renvoyée, même si vous pouvez facilement modifier les méthodes pour renvoyer des exceptions. Troisièmement, la classe ne peut pas empêcher l'utilisation de l'opérateur d'accès au tableau pour insérer des valeurs de n'importe quel type dans le tableau. Quatrièmement, le style de codage privilégie la simplicité par rapport à l'optimisation des performances.

Remarque : *vous pouvez utiliser la technique décrite ici pour créer un tableau typé. Cependant, utiliser un objet Vector constitue une meilleure démarche. Une occurrence de Vector est un véritable tableau typé. Elle dépasse la classe Array ou toute sous-classe par ses performances et ses améliorations. Une illustration de la création d'une sous-classe Array constitue l'objet de cette description.*

Déclaration de la sous-classe

Utilisez le mot-clé `extends` pour indiquer qu'une classe est une sous-classe d'Array. Une sous-classe d'Array doit utiliser l'attribut `dynamic`, comme la classe Array. Autrement, votre sous-classe ne fonctionne pas correctement.

Le code suivant représente la définition de la classe TypedArray, qui comporte une constante contenant le type de données, une méthode de constructeur et les quatre méthodes permettant d'ajouter des éléments au tableau. Le code pour chaque méthode est omis dans cet exemple, mais il est décrit de façon détaillée dans les sections qui suivent :

```
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 {}
}
```

Les quatre méthodes de remplacement utilisent l'espace de noms AS3 au lieu de l'attribut `public` car cet exemple suppose que l'option `-as3` du compilateur est définie sur `true` et l'option `-es` du compilateur sur `false`. Il s'agit des paramètres par défaut pour Adobe Flex Builder 3 et Adobe® Flash® CS4 Professional. Pour plus d'informations, consultez la section « Espace de noms d'ActionScript 3 » à la page 125.

Si vous êtes un développeur expérimenté et que vous préférez utiliser l'héritage de prototype, vous pouvez apporter deux changements mineurs à la classe TypedArray afin qu'elle compile avec l'option `-es` du compilateur définie sur `true`. Commencez par supprimer toutes les occurrences de l'attribut `override` et remplacez l'espace de noms AS3 par l'attribut `public`. Remplacez ensuite `Array.prototype` pour les quatre occurrences de `super`.

Constructeur TypedArray

Le constructeur de sous-classe pose un défi intéressant car il doit accepter une liste d'arguments de longueur arbitraire. Il s'agit de savoir comment transférer les arguments au superconstructeur pour créer le tableau. Si vous transmettez la liste des arguments sous forme d'un tableau, le superconstructeur le considère comme un seul argument de type Array et le tableau résultant a toujours une longueur d'1 élément. Le transfert de listes d'arguments se fait généralement au moyen de la méthode `Function.apply()`, qui prend un tableau d'arguments comme second paramètre mais le convertit en une liste d'arguments lors de l'exécution de la fonction. Malheureusement, vous ne pouvez pas utiliser la méthode `Function.apply()` avec des constructeurs.

La seule solution est de recréer la logique du constructeur Array dans le constructeur TypedArray. Le code suivant indique l'algorithme utilisé dans le constructeur de classe Array, que vous pouvez réutiliser dans votre constructeur de sous-classe Array :

```
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]
            }
        }
    }
}
```

Le constructeur TypedArray partage une grande partie du code du constructeur Array, avec seulement quatre changements apportés au code. Premièrement, la liste des paramètres comprend un nouveau paramètre obligatoire de type Class qui permet d'indiquer le type de données du tableau. Deuxièmement, le type de données transmis au constructeur est affecté à la variable `dataType`. Troisièmement, dans l'instruction `else`, la valeur de la propriété `length` est affectée après la boucle `for` de façon à ce que `length` comprenne uniquement des arguments du type correct. Quatrièmement, le corps de la boucle `for` utilise la version de remplacement de la méthode `push()` de façon à ce que seuls des arguments du type de données correct soient ajoutés au tableau. L'exemple suivant présente la fonction constructeur 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;
        }
    }
}
```

Méthodes de remplacement TypedArray

La classe TypedArray remplace les quatre méthodes de la classe Array qui permettent d'ajouter des éléments à un tableau. Dans chaque cas, la méthode de remplacement ajoute une vérification du type qui empêche d'ajouter des éléments qui ne sont pas du type de données correct. Chaque méthode appelle ensuite sa version de superclasse.

La méthode `push()` parcourt en boucle la liste des arguments avec une boucle `for..in` et effectue une vérification du type sur chaque argument. Les arguments qui ne sont pas de type correct sont supprimés du tableau `args` avec la méthode `splice()`. Une fois que la boucle `for..in` se termine, le tableau `args` contient des valeurs de type `dataType` uniquement. La version de superclasse de `push()` est ensuite appelée avec le tableau `args` mis à jour, comme indiqué dans le code suivant :

```
    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));
    }
```

La méthode `concat()` crée un TypedArray temporaire appelé `passArgs` pour stocker les arguments soumis à la vérification de type. Ceci permet de réutiliser le code de vérification de type qui existe dans la méthode `push()`. Une boucle `for..in` effectue une itération sur le tableau `args` et appelle `push()` sur chaque argument. Etant donné que `passArgs` est typé sous la forme TypedArray, la version TypedArray de `push()` est exécutée. La méthode `concat()` appelle ensuite sa version de superclasse, comme indiqué dans le code suivant :

```
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));
}
```

La méthode `splice()` prend une liste d'arguments arbitraire, mais les deux premiers arguments se réfèrent toujours à un numéro d'index et au nombre d'éléments à supprimer. C'est pourquoi la méthode de remplacement `splice()` effectue la vérification de type uniquement pour les éléments du tableau `args` dans les positions d'index 2 ou supérieures. Il est intéressant de noter que dans le code, il semble y avoir un appel récursif à `splice()` à l'intérieur de la boucle `for`, mais en réalité, ce n'est pas le cas car `args` est de type Array et non de type TypedArray, ce qui signifie que l'appel à `args.splice()` est un appel à la version de superclasse de la méthode. Une fois que la boucle `for..in` se termine, le tableau `args` contient des valeurs du type correct uniquement dans les positions d'index 2 ou supérieures, et `splice()` appelle sa version de superclasse, comme indiqué dans le code suivant :

```
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));
}
```

La méthode `unshift()`, qui ajoute des éléments au début d'un tableau, accepte une liste d'arguments arbitraire également. La méthode de remplacement `unshift()` utilise un algorithme très semblable à celui utilisé par la méthode `push()`, comme indiqué dans le code suivant :

```
    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));
    }
}
```

Exemple : PlayList

L'exemple PlayList présente les techniques d'utilisation des tableaux, dans le contexte d'une application de lecture musicale qui gère une liste de chansons. Ces techniques sont les suivantes :

- Création d'un tableau indexé
- Ajout d'éléments à un tableau indexé
- Tri d'un tableau d'objets en fonction de différentes propriétés, à l'aide d'options de tri différentes
- Conversion d'un tableau en une chaîne séparée par des caractères

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application PlayList se trouvent dans le dossier Samples/PlayList. L'application se compose des fichiers suivants :

Fichier	Description
PlayList.mxml ou PlayList.fla	Le fichier d'application principal dans Flash (FLA) ou Flex (MXML).
com/example/programmingas3/playlist/PlayList.as	Classe représentant une liste de morceaux. Elle utilise un tableau pour enregistrer la liste et gère le tri des éléments de la liste.
com/example/programmingas3/playlist/Song.as	Objet de valeur représentant des informations sur une seule chanson. Les éléments gérés par la classe PlayList sont des occurrences Song.
com/example/programmingas3/playlist/SortProperty.as	Pseudo-énumération dont les valeurs disponibles représentent les propriétés de la classe Song en fonction desquelles une liste d'objets Song peut être triée.

Présentation de la classe PlayList

La classe PlayList gère un ensemble d'objets Song. Elle a des méthodes publiques qui permettent d'ajouter une chanson à la liste de lecture (la méthode `addSong()`) et de trier les chansons dans la liste (la méthode `sortList()`). En outre, la classe comprend une propriété d'accesseur en lecture seule, `songList`, qui permet d'accéder au groupe de chansons dans la liste de lecture. En interne, la classe PlayList conserve une trace de ses chansons à l'aide d'une variable Array privée :

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

En plus de la variable Array `_songs` utilisée par la classe PlayList pour conserver une trace de sa liste de chansons, deux autres variables privées vérifient si la liste doit être triée (`_needToSort`) et contrôlent la propriété sur laquelle est basé le tri de la liste de chansons à un moment donné (`_currentSort`).

Comme avec tous les objets, lorsque vous avez déclaré une occurrence de Array, vous n'avez effectué que la moitié du travail consistant à créer un tableau. Avant d'accéder à des méthodes ou à des propriétés d'une occurrence de Array, cette dernière doit être instanciée dans le constructeur de la classe PlayList.

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

La première ligne du constructeur instancie la variable `_songs` pour qu'elle puisse être utilisée. En outre, la méthode `sortList()` est appelée pour définir la propriété de tri initiale.

Ajout d'une chanson à la liste

Lorsqu'un utilisateur ajoute une nouvelle chanson dans l'application, le code dans le formulaire de saisie des données appelle la méthode `addSong()` de la classe PlayList.

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

A l'intérieur de `addSong()`, la méthode `push()` du tableau `_songs` est appelée. Ceci permet d'ajouter l'objet Song transmis à `addSong()` en tant que nouvel élément dans ce tableau. Avec la méthode `push()` , le nouvel élément est ajouté à la fin du tableau, indépendamment du tri appliqué précédemment. Ceci signifie qu'une fois que la méthode `push()` a été appelée, la liste des chansons risque de ne plus être triée correctement. Par conséquent, la variable `_needToSort` est définie sur `true`. Théoriquement, la méthode `sortList()` pourrait être appelée immédiatement afin d'éviter de vérifier si la liste est triée ou non à un moment donné. En pratique, cependant, la liste des chansons n'a pas besoin d'être triée jusqu'au moment précédant immédiatement sa récupération. En retardant l'opération de tri, l'application n'effectue pas de tri inutile si, par exemple, plusieurs chansons sont ajoutées à la liste avant sa récupération.

Tri de la liste de chansons

Etant donné que les occurrences Song gérées par la liste de lecture sont des objets complexes, les utilisateurs de l'application peuvent trier la liste de lecture en fonction de différentes propriétés (titre de la chanson ou année de publication, par exemple). Dans l'application PlayList, le tri de la liste des chansons s'effectue en trois étapes : identification de la propriété sur laquelle est basé le tri de la liste, indication des options de tri à utiliser lors du tri en fonction de cette propriété et exécution du tri.

Propriétés de tri

Un objet Song conserve la trace de plusieurs propriétés, notamment le titre de la chanson, l'artiste, l'année de publication, le nom du fichier et un ensemble de genres sélectionné par l'utilisateur auquel la chanson appartient. Seules les trois premières propriétés sont pratiques pour le tri. Dans un souci de commodité pour les développeurs, l'exemple inclut la classe SortProperty, qui agit comme une énumération avec des valeurs représentant les propriétés disponibles pour le tri.

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

La classe SortProperty contient trois classes, `TITLE`, `ARTIST` et `YEAR`. Chacune d'elles stocke une chaîne comportant le nom de la propriété de la classe Song pouvant être utilisée pour le tri. Chaque fois qu'une propriété de tri est indiquée dans le reste du code, le membre de l'énumération est utilisé. Par exemple, dans le constructeur PlayList, la liste est triée initialement en appelant la méthode `sortList()`, comme suit :

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

Etant donné que la propriété de tri est spécifiée sous la forme `SortProperty.TITLE`, les chansons sont triées par titre.

Tri par propriété et définition d'options de tri

La classe PlayList trie la liste de chansons dans la méthode `sortList()`, comme suit :

```
/**
 * 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;
}
```

Lors du tri par titre ou par artiste, il est préférable d'effectuer un tri par ordre alphabétique. En revanche, lors du tri par année, il est plus logique d'effectuer un tri numérique. L'instruction `switch` sert à définir l'option de tri appropriée, stockée dans la variable `sortOptions`, en fonction de la valeur indiquée dans le paramètre `sortProperty`. Ici encore, les membres de l'énumération nommés sont utilisés pour faire la différence entre les propriétés, plutôt que les valeurs absolues.

Une fois que vous avez déterminé la propriété et les options de tri, le tableau `_songs` est trié en appelant sa méthode `sortOn()`, en transmettant ces deux valeurs comme paramètres. La propriété de tri est enregistrée et la liste des chansons est triée.

Combinaison d'éléments de tableau en une chaîne séparée par des caractères

Cet exemple utilise non seulement un tableau pour conserver la liste des chansons dans la classe PlayList mais également des tableaux dans la classe Song pour gérer la liste des genres auxquels une chanson appartient. Considérons ce fragment de code issu de la définition de la classe 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(";");
}
```

Lors de la création d'une occurrence de Song, le paramètre `genres` utilisé pour spécifier le genre (ou les genres) auquel la chanson appartient est défini comme occurrence de Array. Ainsi, vous pouvez regrouper plusieurs genres en une seule variable qui peut être transmise au constructeur. Néanmoins, la classe Song conserve, en interne, les genres dans la variable privée `_genres` sous la forme d'une occurrence de String séparée par des points-virgules. Le paramètre Array est converti en une chaîne séparée par des points-virgules en appelant sa méthode `join()` avec la valeur de chaîne littérale `";"` comme séparateur spécifié.

De la même façon, les accesseurs `genres` permettent de définir ou de récupérer des genres sous la forme d'un tableau :

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

L'accesseur `genresset` se comporte exactement comme le constructeur `; il accepte un tableau et appelle la méthode` join() pour la convertir en une chaîne séparée par des points-virgules. L'accesseur `get` effectue l'opération inverse : la méthode `split()` de la variable `_genres` est appelée. Elle divise la chaîne en un tableau de valeurs utilisant le séparateur spécifié (la valeur de chaîne littérale `";"` comme précédemment).

Chapitre 9 : Gestion des erreurs

Gérer une erreur signifie insérer du code dans votre application permettant de réagir à une erreur ou de la réparer lors de la compilation ou de l'exécution de cette application. Lorsque votre application gère des erreurs, *quelque chose* se produit en réponse à l'erreur. Dans certaines situations, il se peut en revanche qu'une erreur soit ignorée (échec silencieux) et qu'aucune réponse ne soit fournie. La gestion des erreurs, lorsqu'elle est utilisée correctement, protège votre application et ses utilisateurs contre un comportement inattendu.

Cependant, la gestion des erreurs est une catégorie large qui englobe la réponse à de nombreux types d'erreurs générées lors de la compilation ou de l'exécution. Ce chapitre est consacré à la gestion des erreurs d'exécution, aux différents types d'erreurs susceptibles d'être générées et aux avantages du nouveau système de gestion des erreurs d'ActionScript 3.0. Il explique également comment implémenter des stratégies de gestion des erreurs personnalisées adaptées à vos applications.

Principes de base de la gestion des erreurs

Introduction à la gestion des erreurs

Une erreur d'exécution est une erreur qui se produit dans votre code ActionScript et qui empêche le contenu ActionScript de s'exécuter dans Adobe® Flash® Player ou Adobe® AIR™. Pour vous assurer que votre code ActionScript s'exécute correctement, vous devez écrire le code dans l'application qui gère l'erreur, la répare, la contourne ou informe au moins l'utilisateur qu'elle a eu lieu. Ce processus est appelé *gestion des erreurs*.

La gestion des erreurs est une catégorie large qui englobe la réponse à de nombreux types d'erreurs générées lors de la compilation ou de l'exécution. Les erreurs qui se produisent lors de la compilation sont souvent plus faciles à identifier - vous devez les corriger pour terminer la création d'un fichier SWF. Ce chapitre ne passe pas en revue les erreurs de compilation. Pour plus d'informations sur la rédaction de code sans erreurs de compilation, consultez les chapitres « Syntaxe et langage ActionScript » à la page 38 et « Programmation orientée objets en ActionScript » à la page 93. Ce chapitre se concentre sur les erreurs d'exécution.

Les erreurs d'exécution peuvent être difficiles à détecter car elles se produisent lorsque le code erroné est exécuté. Si un segment de votre programme contient plusieurs branches de code, telle une instruction `if..then..else`, vous devez tester toutes les conditions possibles, avec toutes les valeurs en entrée susceptibles d'être utilisées par un utilisateur réel, pour confirmer que votre code ne contient pas d'erreur.

Les erreurs d'exécution peuvent être divisées en deux catégories : les *erreurs de programme* sont des erreurs dans votre code ActionScript (spécification du type de données incorrect pour un paramètre de méthode, par exemple) ; les *erreurs logiques* sont des erreurs dans la logique (le contrôle des données et la manipulation des valeurs) de votre programme (utilisation de la formule incorrecte pour calculer les taux d'intérêt dans une application bancaire, par exemple). Encore une fois, ces deux types d'erreurs peuvent souvent être détectés et corrigés à l'avance en testant attentivement votre application.

Il serait idéal d'identifier et de supprimer toutes les erreurs de votre application avant de la mettre à la disposition des utilisateurs finaux. Cependant, toutes les erreurs ne peuvent pas être prévues ni évitées. Par exemple, supposez que votre application ActionScript charge des informations depuis un site Web particulier sur lequel vous n'avez aucun contrôle. Si ce site Web n'est pas disponible, la partie de votre application qui dépend de ces données externes ne se comportera pas correctement. L'aspect le plus important de la gestion des erreurs implique une préparation pour ces cas inconnus, ainsi que leur gestion afin de permettre aux utilisateurs de continuer à utiliser votre application ou, tout du moins, de recevoir un message d'erreur convivial expliquant la raison du problème.

Les erreurs d'exécution sont représentées de deux façons dans ActionScript :

- Classes d'erreur : de nombreuses erreurs sont associées à une classe Error. Lorsqu'une erreur se produit, Flash Player ou Adobe AIR crée une occurrence de la classe Error spécifique associée à cette erreur. Votre code peut utiliser les informations contenues dans cet objet erreur pour donner une réponse appropriée à l'erreur.
- Evénements d'erreur : il arrive qu'une erreur se produise lorsque Flash Player ou Adobe AIR déclencherait normalement un événement. Si tel est le cas, Flash Player et Adobe AIR déclenchent alors un événement d'erreur. A l'instar d'autres événements, chaque événement d'erreur possède une classe qui lui est associée, et Flash Player et Adobe AIR transmettent une occurrence de cette classe aux méthodes qui sont enregistrées auprès de l'événement d'erreur.

Pour déterminer si une méthode particulière peut déclencher une erreur ou un événement d'erreur, consultez l'entrée de la méthode dans le Guide de référence du langage et des composants ActionScript 3.0.

Tâches courantes de gestion des erreurs

Exemples de tâches courantes liées à des erreurs que vous effectuez avec votre code :

- Ecriture de code pour gérer des erreurs
- Test, capture et renvoi d'erreurs
- Définition de votre classe d'erreur
- Réponse à des événements d'erreurs et de statut

Concepts importants et terminologie

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Asynchrone : commande de programme telle qu'un appel de méthode qui ne fournit pas un résultat immédiat, mais qui produit un résultat (ou une erreur) sous la forme d'un événement.
- Capture : lorsqu'une exception (une erreur d'exécution) se produit et que votre code la découvre, ce dernier la *capture*. Lorsqu'une exception est capturée, Flash Player et Adobe AIR cessent d'indiquer à un autre code ActionScript que l'exception s'est produite.
- Version de débogage : version spéciale de Flash Player ou Adobe AIR (ADL), qui contient le code requis pour avertir les utilisateurs de la présence d'erreurs d'exécution. Dans la version standard de Flash Player ou Adobe AIR (celle que possèdent la plupart des utilisateurs), les erreurs qui ne sont pas gérées par votre code ActionScript sont ignorées. Dans les versions de débogage (qui sont incluses avec Adobe Flash CS4 Professional et Adobe Flex), un message d'avertissement apparaît lorsqu'une erreur non gérée se produit.
- Exception : erreur qui se produit lorsqu'un programme est exécuté et que l'environnement d'exécution (Flash Player ou Adobe AIR) ne peut pas résoudre.
- Renvoi : lorsque votre code capture une exception, Flash Player et Adobe AIR cessent de signaler l'exception à d'autres objets. S'il est important pour d'autres objets que l'exception leur soit signalée, votre code doit *renvoyer* l'exception pour recommencer le processus de notification.

- Synchrone : commande de programme (un appel de méthode, par exemple) qui fournit un résultat immédiat (ou qui renvoie immédiatement une erreur), ce qui signifie que la réponse peut être utilisée dans le même bloc de code.
- Envoi : le fait de signaler à Flash Player ou Adobe AIR (et par conséquent, à d'autres objets et au code ActionScript) qu'une erreur s'est produite s'appelle *envoyer* une erreur.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester des exemples de code. Tous les codes de ce chapitre comprennent l'appel de la fonction `trace()`. Pour tester les codes de ce chapitre :

1. Créez un document Flash vide.
2. Sélectionnez une image-clé dans le scénario.
3. Ouvrez le panneau Actions et copiez le code dans le panneau Script.
4. Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

   Les résultats des fonctions `trace()` des codes s'affichent dans le panneau Sortie.

Certains des prochains codes sont plus complexes et sont écrits sous la forme d'une classe. Pour tester ces exemples :

1. Créez un document Flash vide et enregistrez-le sur votre ordinateur.
2. Créez un nouveau fichier ActionScript et enregistrez-le dans le même répertoire que le document Flash. Le nom du fichier doit correspondre au nom de la classe du code. Par exemple, si le code définit une classe ErrorTest, enregistrez le fichier ActionScript sous le nom ErrorTest.as.
3. Copiez le code dans le fichier ActionScript et enregistrez le fichier.
4. Dans le document Flash, cliquez sur une partie vide de la scène ou de l'espace de travail pour activer l'Inspecteur des Propriétés du document.
5. Dans l'Inspecteur des Propriétés, dans le champ Classe du document, saisissez le nom de la classe ActionScript que vous avez copiée du texte.
6. Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

   Les résultats de l'exemple s'affichent dans le panneau Sortie (si l'exemple utilise la fonction `trace()`) ou dans un champ de texte créé par l'exemple de code.

Ces techniques de test d'exemples de code sont décrites de manière plus détaillée dans « Test des exemples de code contenus dans un chapitre » à la page 36.

Types d'erreurs

Lorsque vous développez et exécutez des applications, vous rencontrez différents types d'erreurs et de termes. La liste suivante présente les principaux termes et types d'erreurs :

- *Erreurs de compilation* : générées par le compilateur ActionScript lors de la compilation du code. Les erreurs de compilation ont lieu lorsque des problèmes de syntaxe dans votre code empêchent de créer votre application.

- *Erreurs d'exécution* : générées lorsque vous exécutez votre application après l'avoir compilée. Les erreurs d'exécution représentent des erreurs qui se produisent lors de la lecture d'un fichier SWF dans Adobe Flash Player ou Adobe AIR. Dans la plupart des cas, il est possible de gérer les erreurs d'exécution au moment où elles se produisent, de les signaler à l'utilisateur et de prendre les mesures requises pour poursuivre l'exécution de votre application. S'il s'agit d'une erreur grave (impossibilité de se connecter à un site Web distant ou de charger des données), vous pouvez utiliser la gestion des erreurs pour mettre fin à votre application en douceur.
- *Erreurs synchrones* : générées lorsqu'une fonction est appelée—par exemple, lorsque vous tentez d'utiliser une méthode spécifique et que l'argument que vous lui transmettez n'est pas valide, Flash Player ou Adobe AIR renvoie une exception. La plupart des erreurs se produisent en mode synchrone (au moment de l'exécution d'une instruction) et le flux de contrôle passe immédiatement à l'instruction `catch` la plus appropriée.

  Par exemple, l'extrait de code suivant renvoie une erreur d'exécution, car la méthode `browse()` n'est pas appelée avant que le programme ne tente de charger un fichier :

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

  Dans ce cas, une erreur d'exécution est renvoyée de façon synchrone car Flash Player a déterminé que la méthode `browse()` n'a pas été appelée avant la tentative de chargement du fichier.

  Pour obtenir des informations détaillées relatives à la gestion des erreurs synchrones, consultez la section « Gestion des erreurs synchrones dans une application » à la page 193.

- *Erreursasynchrones* : générées à différents moments lors de l'exécution. Elles provoquent des événements et sont interceptées par des écouteurs d'événement. Une opération asynchrone est une opération dans laquelle une fonction lance une opération mais n'attend pas qu'elle se termine. Vous pouvez créer un écouteur d'événement d'erreur pour attendre que l'application ou l'utilisateur tente une opération, et si cette dernière échoue, vous interceptez l'erreur avec un écouteur d'événement et répondez à l'événement d'erreur. Ensuite, l'écouteur d'événement appelle une fonction de gestionnaire d'événement pour répondre à l'événement d'erreur avec pertinence. Par exemple, le gestionnaire d'événement peut lancer une boîte de dialogue qui invite l'utilisateur à résoudre l'erreur.

  Reprenez l'exemple d'erreur synchrone lors du chargement d'un fichier présenté précédemment. Si vous réussissez à appeler la méthode `browse()` avant de lancer le chargement d'un fichier, Flash Player distribue plusieurs événements. Par exemple, au démarrage d'un chargement, l'événement `open` est distribué. A la fin du chargement, l'événement `complete` est distribué. Etant donné que la gestion d'événements est asynchrone (c'est-à-dire qu'elle n'a pas lieu à des moments prédéfinis, connus et spécifiques), vous devez utiliser la méthode `addEventListener()` pour détecter ces événements spécifiques, comme l'indique le code suivant :

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.OPEN, openHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();

function selectHandler(event:Event):void
{
    trace("...select...");
    var request:URLRequest = new URLRequest("http://www.yourdomain.com/fileupload.cfm");
    request.method = URLRequestMethod.POST;
    event.target.upload(request.url);
}
function openHandler(event:Event):void
{
    trace("...open...");
}
function completeHandler(event:Event):void
{
    trace("...complete...");
}
```

Pour obtenir des informations détaillées sur la gestion des erreurs asynchrones, consultez la section « Réponse à des événements et au statut d'erreur » à la page 198.

- *Exceptions non interceptées* : renvoyées sans logique correspondante (telle une instruction `catch`) pour y répondre. Si votre application renvoie une erreur, et qu'aucune instruction `catch` ni gestionnaire d'événement approprié n'est trouvé au niveau actuel ou supérieur pour gérer l'erreur, cette dernière est considérée comme une exception non interceptée.

  Lors de l'exécution, Flash Player ignore les erreurs non interceptées et tente de poursuivre la lecture si l'erreur n'arrête pas le fichier SWF actuel, car les utilisateurs ne peuvent pas nécessairement résoudre une erreur eux-mêmes. Le fait d'ignorer une erreur non interceptée est appelé échec silencieux et peut compliquer le débogage des applications. La version de débogage de Flash Player répond à une erreur non interceptée en terminant le script courant et en affichant l'erreur non interceptée dans le résultat de l'instruction `trace` ou en enregistrant le message d'erreur dans un fichier journal. Si l'objet d'exception est une occurrence de la classe Error ou l'une de ses sous-classes, la méthode `getStackTrace()` est appelée, et les informations de trace de pile sont également affichées dans le résultat de l'instruction trace ou dans un fichier journal. Pour plus d'informations sur l'utilisation de la version de débogage de Flash Player, consultez la section « Utilisation des versions de débogage de Flash Player et AIR » à la page 192.

Gestion des erreurs dans ActionScript 3.0

Etant donné que de nombreuses applications peuvent être exécutées sans créer de logique pour gérer les erreurs, les développeurs sont tentés de retarder la création de la gestion des erreurs dans leurs applications. Néanmoins, sans gestion des erreurs, une application risque de s'interrompre facilement ou de poser des problèmes à l'utilisateur si quelque chose ne fonctionne pas comme prévu. ActionScript 2.0 possède une classe Error qui vous permet de créer une logique dans des fonctions personnalisées afin de renvoyer une exception avec un message spécifique. Etant donné que la gestion des erreurs est cruciale pour rendre une application conviviale, ActionScript 3.0 inclut une architecture étendue pour intercepter les erreurs.

Remarque : *le Guide de référence du langage et des composants ActionScript 3.0 documente les exceptions renvoyées par de nombreuses méthodes, mais risque de ne pas inclure toutes les exceptions associées à chacune d'elles. Une méthode peut renvoyer une exception due à une erreur de syntaxe ou d'autres problèmes qui ne sont pas signalés explicitement dans la description de la méthode, même lorsque cette dernière répertorie les exceptions qu'une méthode renvoie.*

Eléments de gestion des erreurs ActionScript 3.0

ActionScript 3.0 comprend de nombreux outils permettant de gérer les erreurs, notamment :

- Classes Error : ActionScript 3.0 comprend un large éventail de classes Error destiné à développer la portée des situations susceptibles de produire des objets erreur. Chaque classe Error permet aux applications de gérer et de répondre à des conditions d'erreur spécifiques, qu'elles soient liées à des erreurs système (comme une condition MemoryError), à des erreurs de codage (comme une condition ArgumentError), à des erreurs de réseau et de communication (comme une condition URIError), ou d'autres situations. Pour plus d'informations sur chaque classe, consultez la section « Comparaison des classes Error » à la page 201.
- Moins d'échecs silencieux : dans les versions précédentes de Flash Player, les erreurs étaient générées et signalées uniquement si vous utilisiez explicitement l'instruction `throw`. Pour Flash Player 9 et les versions ultérieures ainsi qu'Adobe AIR, les propriétés et les méthodes ActionScript natives renvoient des erreurs qui vous permettent de gérer ces exceptions de façon plus efficace lorsqu'elles se produisent, puis de réagir individuellement à chaque exception.
- Messages d'erreur clairs affichés lors du débogage : lorsque vous utilisez la version de débogage de Flash Player ou Adobe AIR, les situations ou le code problématiques génèrent des messages d'erreur détaillés qui vous aident à identifier les raisons de l'échec d'un bloc de code particulier. Ceci permet de réparer les erreurs de façon plus efficace. Pour plus d'informations, consultez la section « Utilisation des versions de débogage de Flash Player et AIR » à la page 192.
- Les erreurs précises permettent d'afficher des messages d'erreur clairs pour les utilisateurs lors de l'exécution. Dans les versions précédentes de Flash Player, la méthode `FileReference.upload()` renvoyait la valeur booléenne `false` en cas d'échec de l'appel `upload()`, indiquant l'une des cinq erreurs possibles. Si une erreur se produit lorsque vous appelez la méthode `upload()` dans ActionScript 3.0, vous pouvez renvoyer l'une des quatre erreurs spécifiques afin d'afficher des messages d'erreur plus précis pour les utilisateurs finaux.
- Gestion des erreurs affinée : des erreurs distinctes sont renvoyées pour de nombreuses situations courantes. Par exemple, dans ActionScript 2.0, avant qu'un objet FileReference ne soit renseigné, la propriété `name` a la valeur `null` (par conséquent, avant d'utiliser ou d'afficher la propriété `name`, vous devez vérifier que la valeur est définie et qu'elle n'est pas `null`). Dans ActionScript 3.0, si vous tentez d'accéder à la propriété `name` avant qu'elle ne soit renseignée, Flash Player ou AIR renvoie une erreur IllegalOperationError qui vous indique que la valeur n'a pas été définie. Vous pouvez utiliser des blocs `try..catch..finally` pour gérer l'erreur. Pour plus d'informations, consultez la section « Utilisation des instructions try..catch..finally » à la page 193.
- Aucun problème sérieux de performance : l'utilisation de blocs `try..catch..finally` pour gérer des erreurs ne nécessite pas ou peu de ressources supplémentaires par rapport aux versions précédentes d'ActionScript.
- Une classe ErrorEvent qui vous permet de créer des écouteurs pour des événements d'erreurs asynchrones spécifiques : pour plus d'informations, consultez la section « Réponse à des événements et au statut d'erreur » à la page 198.

Stratégies de gestion des erreurs

Tant que votre application ne rencontre pas de condition problématique, vous pouvez continuer à l'exécuter sans créer de logique de gestion des erreurs dans votre code. En revanche, si vous ne gérez pas d'erreurs de façon active et que votre application rencontre un problème, vos utilisateurs ignoreront toujours la raison de son échec.

Vous pouvez aborder la gestion des erreurs de diverses façons dans votre application. La liste suivante résume les trois principales options de gestion des erreurs :

- Utilisez les instructions `try..catch..finally`. Elles interceptent les erreurs synchrones lorsqu'elles se produisent. Vous pouvez imbriquer vos instructions dans une hiérarchie pour intercepter des exceptions à différents niveaux d'exécution du code. Pour plus d'informations, consultez la section « Utilisation des instructions try..catch..finally » à la page 193.
- Créez des objets d'erreur personnalisés. Vous pouvez utiliser la classe Error pour créer des objets d'erreur personnalisés afin de suivre des opérations spécifiques dans votre application qui ne sont pas couvertes par des types d'erreur intégrés. Vous pouvez ensuite appliquer des instructions `try..catch..finally` aux objets d'erreur personnalisés. Pour plus d'informations, consultez la section « Création de classes d'erreur personnalisées » à la page 197.
- Ecrivez des gestionnaires et des écouteurs d'événement pour répondre à des événements d'erreur. Cette stratégie vous permet de créer des gestionnaires d'erreurs globaux pour gérer des événements identiques sans dupliquer beaucoup de code dans des blocs `try..catch..finally`. Il est également plus probable que vous interceptiez des erreurs asynchrones à l'aide de cette approche. Pour plus d'informations, consultez la section « Réponse à des événements et au statut d'erreur » à la page 198.

Utilisation des versions de débogage de Flash Player et AIR

Adobe propose aux développeurs des éditions spéciales de Flash Player et Adobe AIR, destinées à les aider à exécuter des opérations de débogage. Vous obtenez une copie de la version de débogage de Flash Player lorsque vous installez Adobe Flash CS4 ou Adobe Flex Builder 3. Vous disposez également de la version de débogage d'Adobe AIR, appelée ADL, lorsque vous installez l'un de ces outils ou dans le cadre de l'installation du SDK d'Adobe AIR.

Il existe une grande différence dans la façon dont les versions de débogage et les versions de Flash Player et Adobe AIR mises sur le marché signalent les erreurs. Les versions de débogage indiquent le type d'erreur (Error, IOError ou EOFError générique), le numéro de l'erreur et un message d'erreur sous une forme lisible par une personne. Les versions mises sur le marché indiquent uniquement le type d'erreur et son numéro. Considérons par exemple le code qui suit :

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

Si la méthode `readBoolean()` a renvoyé une erreur EOFError dans la version de débogage de Flash Player, le message suivant s'affiche dans le champ de texte `tf` : « EOFError: Erreur #2030: Fin de fichier détectée ».

Dans une version mise sur le marché de Flash Player ou Adobe AIR, le même code afficherait le texte suivant : « EOFError: Erreur #2030 ».

Il n'existe pas de chaînes de message d'erreur afin de réduire au minimum les ressources et la taille des versions mises sur le marché. Vous pouvez consulter le numéro d'erreur dans la documentation (annexes du Guide de référence du langage et des composants ActionScript 3.0) pour l'associer à un message d'erreur. Vous pouvez également reproduire l'erreur dans les versions de débogage de Flash Player et AIR pour visualiser le message entier.

Gestion des erreurs synchrones dans une application

La gestion des erreurs la plus courante est la logique de gestion des erreurs synchrones, où vous insérez des instructions dans votre code pour intercepter des erreurs synchrones lors de l'exécution. Ce type de gestion des erreurs permet à votre application de repérer des erreurs d'exécution et de les résoudre lorsque des fonctions échouent. La logique d'interception d'une erreur synchrone fait appel aux instructions `try..catch..finally`, qui tentent littéralement une opération, interceptent toute réponse à l'erreur émanant de Flash Player ou Adobe AIR, puis exécutent une autre opération pour gérer l'opération qui a échoué.

Utilisation des instructions try..catch..finally

Lorsque vous manipulez des erreurs d'exécution synchrones, utilisez les instructions `try..catch..finally` pour intercepter les erreurs. Lorsqu'une erreur d'exécution se produit, Flash Player ou Adobe AIR renvoie une exception, ce qui signifie qu'il suspend l'exécution normale et crée un objet spécial de type Error. L'objet Error est ensuite renvoyé au premier bloc `catch` disponible.

L'instruction `try` regroupe les instructions pouvant créer des erreurs. Vous utilisez toujours l'instruction `catch` avec une instruction `try`. Si une erreur est détectée dans l'une des instructions du bloc `try`, les instructions `catch` associées à cette instruction `try` sont exécutées.

L'instruction `finally` regroupe les instructions exécutées, qu'une erreur se produise ou non dans le bloc `try`. S'il ne se produit pas d'erreur, les instructions du bloc `finally` sont exécutées au terme de l'exécution des instructions du bloc `try`. S'il se produit une erreur, l'instruction `catch` appropriée est exécutée en premier lieu, suivie des instructions du bloc `finally`.

Le code suivant illustre la syntaxe d'utilisation des instructions `try..catch..finally` :

```
try
{
    // some code that could throw an error
}
catch (err:Error)
{
    // code to react to the error
}
finally
{
    // Code that runs whether or not an error was thrown. This code can clean
    // up after the error, or take steps to keep the application running.
}
```

Chaque instruction `catch` identifie un type d'exception spécifique qu'elle gère. L'instruction `catch` peut spécifier uniquement des classes d'erreur qui sont des sous-classes de la classe Error. Chaque instruction `catch` est vérifiée dans l'ordre. Seule la première instruction `catch` qui correspond au type d'erreur renvoyé est exécutée. En d'autres termes, si vous vérifiez d'abord la classe Error de niveau supérieur, puis une sous-classe de la classe Error, seule la classe Error de niveau supérieur correspond. Le code suivant illustre ce point :

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

Le code précédent affiche le résultat suivant :

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

Si vous souhaitez intercepter correctement l'erreur ArgumentError, vous devez vérifier que les types d'erreur les plus spécifiques sont répertoriés en premier, suivis des types d'erreur les plus génériques, comme l'indique le code suivant :

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

Plusieurs méthodes et propriétés dans l'API de Flash Player renvoient des erreurs d'exécution si elles en rencontrent lors de leur exécution. Par exemple, la méthode `close()` de la classe Sound renvoie une erreur IOError si la méthode ne parvient pas à fermer le flux audio, comme indiqué dans le code suivant :

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

Au fur et à mesure que vous vous familiariserez avec le Guide de référence du langage et des composants ActionScript 3.0, vous identifierez les méthodes qui renvoient des exceptions, comme indiqué dans la description de chaque méthode.

Instruction throw

Flash Player et Adobe AIR renvoient des exceptions s'ils rencontrent des erreurs lors de l'exécution de votre application. En outre, vous pouvez renvoyer des exceptions de façon explicite à l'aide de l'instruction `throw`. Si tel est le cas, Adobe vous conseille de renvoyer des occurrences de la classe Error ou de ses sous-classes. Le code suivant illustre une instruction `throw` qui renvoie une occurrence de la classe Error, `MyErr`, et appelle une fonction, `myFunction()` en réponse au renvoi de l'erreur :

```
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.
}
```

Les instructions `catch` sont classées de façon à ce que les types de données les plus spécifiques apparaissent en premier. Si l'instruction `catch` pour le type de données Number est répertoriée en premier, ni l'instruction `catch` pour le type de données uint, ni l'instruction catch pour le type de données int n'est exécutée.

Remarque : *dans le langage de programmation Java, chaque fonction qui peut renvoyer une exception doit le déclarer en répertoriant les classes d'exception qu'elle peut renvoyer dans une clause `throws` associée à la déclaration de la fonction. ActionScript n'exige pas que vous déclariez les exceptions pouvant être renvoyées par une fonction.*

Affichage d'un message d'erreur simple

L'un des avantages majeurs du nouveau modèle d'événement d'erreur et d'exception consiste à permettre d'informer les utilisateurs du moment où une action échoue et de la raison de cet échec. Votre rôle consiste à écrire le code pour afficher le message et à offrir des options en réponse.

Le code suivant illustre une instruction `try..catch` simple permettant d'afficher l'erreur dans un champ de texte :

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

En utilisant un plus grand nombre de classes d'erreur et d'erreurs de compilateur intégrées, ActionScript 3.0 fournit de plus amples informations sur les raisons de l'échec d'une action que les versions précédentes. Ceci vous permet de créer des applications plus stables avec une meilleure gestion des erreurs.

Renvoi des erreurs

Lorsque vous créez des applications, il peut souvent arriver que vous soyez amené à renvoyer une erreur si vous ne parvenez pas à la gérer correctement. Par exemple, le code suivant illustre un bloc `try..catch` imbriqué, qui renvoie une erreur ApplicationError personnalisée si le bloc `catch` imbriqué n'est pas capable de gérer l'erreur :

```
try
{
    try
    {
        trace("<< try >>");
        throw new ArgumentError("some error which will be rethrown");
    }
    catch (error:ApplicationError)
    {
        trace("<< catch >> " + error);
        trace("<< throw >>");
        throw error;
    }
    catch (error:Error)
    {
        trace("<< Error >> " + error);
    }
}
catch (error:ApplicationError)
{
    trace("<< catch >> " + error);
}
```

Le résultat issu du fragment de code précédent serait le suivant :

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

Le bloc `try` imbriqué renvoie une erreur ApplicationError personnalisée qui est interceptée par le bloc `catch` suivant. Ce bloc `catch` imbriqué peut tenter de gérer l'erreur et, si la tentative échoue, renvoyer l'objet ApplicationError au bloc `try..catch`.

Création de classes d'erreur personnalisées

Vous pouvez étendre l'une des classes Error standard pour créer vos classes d'erreur spécialisées dans ActionScript. Vous pouvez créer vos classes d'erreur pour les motifs suivants :

- Identifier des erreurs ou des groupes d'erreurs spécifiques uniques pour votre application.

  Par exemple, vous pouvez gérer différemment les erreurs renvoyées par votre code, en plus de celles interceptées par Flash Player ou Adobe AIR. Vous pouvez créer une sous-classe de la classe Error pour suivre le nouveau type de données d'erreur dans les blocs `try..catch`.

- Fournir des fonctionnalités d'affichage d'erreurs exceptionnelles pour les erreurs générées par votre application.

  Par exemple, vous pouvez créer une méthode `toString()` qui formate vos messages d'erreur d'une certaine façon. Vous pouvez également définir une méthode `lookupErrorString()` qui prend un code d'erreur et récupère le message adéquat en fonction du langage que l'utilisateur préfère.

Une classe d'erreur spécialisée doit étendre la classe Error d'ActionScript de base. Voici un exemple de classe AppError spécialisée qui étend la classe Error :

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

L'exemple suivant illustre l'utilisation d'une classe AppError dans votre projet :

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

Remarque : *si vous souhaitez remplacer la méthode `Error.toString()` dans votre sous-classe, vous devez lui fournir un paramètre `...(rest)`. La spécification du langage ECMAScript sur laquelle est basé ActionScript 3.0 définit ainsi la méthode `Error.toString()` et ActionScript 3.0 respecte cette définition à des fins de rétrocompatibilité. Par conséquent, lorsque vous remplacez la méthode `Error.toString()` vous devez faire correspondre les paramètres exactement. Vous ne pouvez pas transmettre de paramètres à votre méthode `toString()` lors de l'exécution car ils sont ignorés.*

Réponse à des événements et au statut d'erreur

L'une des améliorations majeures apportées à la gestion des erreurs dans ActionScript 3.0 est la prise en charge de la gestion des événements d'erreur pour répondre à des erreurs d'exécution asynchrones. (Si vous souhaitez obtenir une définition des erreurs asynchrones, consultez la section « Types d'erreurs » à la page 188.)

Vous pouvez créer des écouteurs d'événement et des gestionnaires d'événements pour répondre aux événements d'erreurs. De nombreuses classes distribuent des événements d'erreurs de la même façon que d'autres événements. Par exemple, une occurrence de la classe XMLSocket distribue normalement trois types d'événements :`Event.CLOSE`, `Event.CONNECT` et `DataEvent.DATA`. Néanmoins, lorsqu'un problème se produit, la classe XMLSocket peut distribuer `IOErrorEvent.IOError` ou `SecurityErrorEvent.SECURITY_ERROR`. Pour plus d'informations sur les écouteurs et les gestionnaires d'événement, consultez le chapitre « Gestion des événements » à la page 254.

Les événements d'erreurs peuvent être classés en deux catégories :

- Evénements d'erreurs qui étendent la classe ErrorEvent

  La classe flash.events.ErrorEvent contient les propriétés et les méthodes permettant de gérer les erreurs d'exécution liées à des opérations de réseau et de communication. Les classes AsyncErrorEvent, IOErrorEvent et SecurityErrorEvent étendent la classe ErrorEvent. Si vous utilisez la version de débogage de Flash Player ou Adobe AIR, une boîte de dialogue vous informe, lors de l'exécution, de la présence d'événements d'erreurs sans fonctions d'écouteur rencontrés par le lecteur.

- Evénements d'erreurs basés sur le statut

Les événements d'erreur basés sur le statut sont liés aux propriétés `netStatus` et `status` des classes de communication et de réseau. Si Flash Player ou Adobe AIR rencontre un problème lors de la lecture ou de l'écriture des données, la valeur des propriétés `netStatus.info.level` ou `status.level` (selon l'objet de classe que vous utilisez) est définie sur la valeur `"error"`. Vous répondez à cette erreur en vérifiant que la propriété `level` contient la valeur `"error"` dans votre fonction de gestionnaire d'événement.

Utilisation d'événements d'erreurs

La classe ErrorEvent et ses sous-classes contiennent des types d'erreurs destinés à gérer les erreurs distribuées par Flash Player et Adobe AIR lors de la lecture ou de l'écriture des données.

L'exemple suivant utilise à la fois une instruction `try..catch` et des gestionnaires d'événement d'erreur pour afficher toute erreur détectée lors de la tentative de lecture d'un fichier local. Vous pouvez ajouter un code de gestion plus élaboré pour proposer des options à l'utilisateur ou gérer l'erreur automatiquement aux endroits indiqués par le commentaire « your error-handling code here » :

```
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
        }
    }
}
```

Utilisation d'événements de changement de statut

Flash Player et Adobe AIR changent dynamiquement la valeur des propriétés `netStatus.info.level` ou `status.level` pour les classes qui prennent en charge la propriété `level`. Les classes qui ont la propriété `netStatus.info.level` sont NetConnection, NetStream et SharedObject. Les classes qui ont la propriété `status.level` sont HTTPStatusEvent, Camera, Microphone et LocalConnection. Vous pouvez écrire une fonction de gestionnaire pour répondre au changement de valeur `level` et suivre les erreurs de communication.

L'exemple suivant utilise une fonction `netStatusHandler()` pour tester la valeur de la propriété `level`. Si la propriété `level` indique qu'une erreur a été rencontrée, le code suit le message « Video stream failed ».

```
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);
        }
    }
}
```

Comparaison des classes Error

ActionScript fournit de nombreuses classes Error prédéfinies. Flash Player et Adobe AIR utilisent un grand nombre de ces classes, mais vous pouvez également utiliser les mêmes classes Error dans votre code. Il existe deux types principaux de classes Error dans ActionScript 3.0 : les classes Error de base d'ActionScript et les classes Error du package flash.error. Les classes Error de base sont définies par la spécification de langage ECMAScript (ECMA-262), version 3. Le package flash.error contient des classes supplémentaires permettant le débogage et le développement d'application ActionScript 3.0.

Classes Error de base ECMAScript

Les classes Error de base ECMAScript comprennent les classes Error, EvalError, RangeError, ReferenceError, SyntaxError, TypeError et URIError. Chacune de ces classes se trouve dans l'espace de noms de niveau supérieur.

Nom de classe	Description	Remarques
Error	Vous pouvez utiliser la classe Error pour renvoyer des exceptions. Il s'agit de la classe de base pour les autres classes d'exception définies dans ECMAScript : EvalError, RangeError, ReferenceError, SyntaxError, TypeError et URIError.	La classe Error sert de classe de base pour toutes les erreurs d'exécution renvoyées par Flash® Player et Adobe® AIR®. Elle est recommandée pour toutes les classes d'erreur personnalisées.
EvalError	Une exception EvalError est renvoyée si des paramètres sont transmis au constructeur de la classe Function ou si le code utilisateur appelle la fonction `eval()`.	Dans ActionScript 3.0, la prise en charge de la fonction `eval()` a été supprimée et les tentatives d'utilisation de la fonction provoquent le renvoi d'une erreur. Les versions précédentes de Flash Player utilisaient la fonction `eval()` pour accéder à des variables, des propriétés, des objets ou des clips par nom.
RangeError	Une exception RangeError est renvoyée si une valeur numérique excède la plage acceptable.	Par exemple, une exception RangeError est renvoyée par la classe Timer si un retard est négatif ou non fini. Elle peut également être renvoyée si vous tentez d'ajouter un objet d'affichage à une profondeur non valide.
ReferenceError	Une exception ReferenceError est renvoyée lorsque vous tentez d'utiliser une référence à une propriété non définie pour un objet scellé (non dynamique). Les versions du compilateur ActionScript antérieures à ActionScript 3.0 ne renvoyaient pas d'erreur lorsque vous tentiez d'accéder à une propriété `non définie`. ActionScript 3.0 renvoie toutefois l'exception ReferenceError dans ce cas de figure.	Les exceptions pour des variables non définies pointent vers des bogues éventuels afin d'améliorer la qualité du logiciel. Cependant, si vous n'avez pas l'habitude d'initialiser vos variables, ce nouveau comportement d'ActionScript risque d'exiger des changements dans vos habitudes de codage.

Nom de classe	Description	Remarques
SyntaxError	Une exception SyntaxError est renvoyée lorsqu'il se produit une erreur d'analyse dans votre code ActionScript. Pour plus d'informations, consultez la section 15.11.6.4 du langage de spécification ECMAScript (ECMA-262), version 3 sur le site www.ecma-international.org/publications/standards/Ecma-262.htm, ainsi que la section 10.3.1 de la spécification ECMAScript pour XML (E4X) (ECMA-357, version 2) sur le site www.ecma-international.org/publications/standards/Ecma-357.htm.	Une exception SyntaxError peut être renvoyée dans les cas suivants : • ActionScript renvoie des exceptions SyntaxError lorsqu'une expression régulière non valide est analysée par la classe RegExp. • ActionScript renvoie des exceptions SyntaxError lorsqu'un fichier XML non valide est analysé par la classe XMLDocument.
TypeError	L'exception TypeError est renvoyée lorsque le type réel d'un opérande ne correspond pas au type prévu. Pour plus d'informations, consultez la section 15.11.6.5 du langage de spécification ECMAScript sur le site www.ecma-international.org/publications/standards/Ecma-262.htm, ainsi que la section 10.3 de la spécification EX4 sur le site www.ecma-international.org/publications/standards/Ecma-357.htm.	Une exception TypeError peut être renvoyée dans les cas suivants : • Un paramètre réel de fonction ou de méthode ne peut pas être forcé à correspondre au type de paramètre formel. • Une valeur est affectée à une variable et ne peut pas être forcée à correspondre au type de la variable. • Le côté droit de l'opérateur `is` ou `instanceof` n'est pas un type valide. • L'utilisation du mot clé `super` n'est pas valide. • Une recherche de propriété donne lieu à plusieurs liaisons, soit un résultat ambigu. • Une méthode est appelée pour un objet incompatible. Par exemple, une exception TypeError est renvoyée si une méthode de la classe RegExp est greffée sur un objet générique, puis appelée.
URIError	L'exception URIError est renvoyée lorsque l'une des fonctions de gestion URI globales est utilisée d'une manière qui n'est pas compatible avec sa définition. Pour plus d'informations, consultez la section 15.11.6.6 de la spécification ECMAScript sur le site www.ecma-international.org/publications/standards/Ecma-262.htm.	Une exception URIError peut être renvoyée dans les cas suivants : Un URI non valide est défini par une fonction API de Flash Player qui s'attend à un URI valide, comme `Socket.connect()`.

Classes Error de base d'ActionScript

Outre les classes Error ECMAScript de base, ActionScript ajoute plusieurs classes associées aux conditions d'erreurs propres à ActionScript et à la fonctionnalité de gestion des erreurs.

Etant donné que ces classes sont des extensions de langage ActionScript de la spécification de langage ECMAScript, version 3 qui pourraient être intégrées à une version future de la spécification, elles sont conservées au niveau supérieur au lieu d'être placées dans un package comme flash.error.

Nom de classe	Description	Remarques
ArgumentError	La classe ArgumentError représente une erreur qui se produit lorsque les valeurs de paramètre fournies lors d'un appel de fonction ne correspondent pas aux paramètres définis pour celle-ci.	Voici des exemples d'erreurs d'argument : • Trop ou trop peu d'arguments sont fournis à une méthode. • Un argument devait être membre d'une énumération, et cela n'a pas été le cas.
SecurityError	L'exception SecurityError est renvoyée lorsqu'une violation de sécurité se produit et que l'accès est refusé.	Voici des exemples d'erreurs de sécurité : • Un accès à une propriété ou un appel de méthode non autorisé est effectué en franchissant les limites du sandbox de sécurité. • Il s'est produit une tentative d'accès à une URL non autorisée par le sandbox de sécurité. • Il s'est produit une tentative de connexion socket sur un port, mais le fichier de régulation socket approprié n'était pas présent. • Il s'est produit une tentative d'accès à la caméra ou au microphone de l'utilisateur et celui-ci a refusé la demande d'accès au périphérique.
VerifyError	Une erreur VerifyError est renvoyée lorsqu'un fichier SWF incorrect ou altéré est détecté.	Lorsqu'un fichier SWF charge un autre fichier SWF, le SWF parent peut intercepter une exception VerifyError générée par le SWF chargé.

Classes Error du package flash.error

Le package flash.error regroupe des classes Error qui font partie de l'API de Flash Player. A la différence des classes Error décrites précédemment, le package flash.error communique les événements d'erreurs propres à Flash Player ou Adobe AIR.

Nom de classe	Description	Remarques
EOFError	Une exception EOFError est émise lors d'une tentative de lecture au-delà de la fin des données disponibles.	Par exemple, une exception EOFError est émise chaque fois qu'une méthode de lecture de l'interface IDataInput est appelée et que les données sont insuffisantes pour répondre à la requête de lecture.
IllegalOperationError	Une exception IllegalOperationError est renvoyée lorsqu'une méthode n'est pas implémentée ou si l'implémentation ne couvre pas l'utilisation actuelle.	Voici quelques exemples d'exceptions d'erreurs liées à des opérations non valides : • Une classe de base, telle que DisplayObjectContainer, propose plus de fonctionnalités qu'une scène ne peut prendre en charge. Par exemple, si vous tentez d'obtenir ou de définir un masque sur la scène (à l'aide de `stage.mask`), Flash Player et Adobe AIR renvoient une exception IllegalOperationError accompagnée du message « La classe Stage n'implémente ni cette propriété, ni cette méthode ». • Une sous-classe hérite d'une méthode dont elle n'a pas besoin et qu'elle ne souhaite pas prendre en charge. • Certaines méthodes d'accessibilité sont appelées lorsque Flash Player est compilé sans les fonctions d'accessibilité. • Les fonctions réservées à la programmation sont appelées à partir d'une version d'exécution de Flash Player. • Vous tentez de définir le nom d'un objet placé sur le scénario.
IOError	Une exception IOError est renvoyée lorsqu'un type d'exception E/S se produit.	Vous obtenez cette erreur, par exemple, lorsque vous tentez une opération de lecture-écriture sur un socket qui n'est pas connecté ou qui est déconnecté.
MemoryError	Une exception MemoryError est renvoyée lors de l'échec d'une requête d'allocation de mémoire.	Par défaut, ActionScript Virtual Machine 2 n'impose pas de limite à la quantité de mémoire qu'un programme ActionScript peut allouer. Sur un PC de bureau, les échecs d'allocation de mémoire ne sont pas fréquents. Une erreur est renvoyée lorsque le système ne parvient pas à allouer la mémoire requise pour une opération. Par conséquent, sur un PC de bureau, cette exception est rare, à moins qu'une requête d'allocation ne soit vraiment importante (par exemple, une requête de 3 milliards d'octets est impossible, car un programme Microsoft® Windows® de 32 bits peut accéder à 2 Go d'espace d'adressage uniquement).
ScriptTimeoutError	Une exception ScriptTimeoutError est renvoyée lorsqu'un intervalle de délai d'expiration du script de 15 secondes est atteint. En interceptant une exception ScriptTimeoutError, vous pouvez gérer le délai d'expiration du script plus en douceur. S'il n'existe aucun gestionnaire d'exceptions, le gestionnaire de l'exception non interceptée affiche une boîte de dialogue contenant un message d'erreur.	Pour éviter qu'un développeur n'intercepte l'exception et reste dans une boucle sans fin, seule la première exception ScriptTimeoutError renvoyée au cours d'un script donné peut être interceptée. Votre code ne peut pas intercepter une exception ScriptTimeoutError ultérieure. Elle passe immédiatement au gestionnaire de l'exception non interceptée.
StackOverflowError	L'exception StackOverflowError est renvoyée lorsque la pile disponible pour le script a été épuisée.	Une exception StackOverflowError peut indiquer qu'un problème de récursivité à l'infini s'est produit.

Exemple : application CustomErrors

L'application CustomErrors décrit les techniques d'utilisation des erreurs personnalisées lors de la création d'une application. Ces techniques sont les suivantes :

- Validation d'un paquet XML
- Ecriture d'une erreur personnalisée
- Renvoi d'erreurs personnalisées
- Notification des utilisateurs lors du renvoi d'une erreur

Pour obtenir les fichiers d'application associés à cet exemple, voir www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application CustomErrors se trouvent dans le dossier Samples/CustomError. L'application se compose des fichiers suivants :

Fichier	Description
CustomErrors.mxml ou CustomErrors.fla	Fichier d'application principal en FLA pour Flash ou en MXML pour Flex
com/example/programmingas3/errors/ApplicationError.as	Une classe servant de classe Error de base pour les classes FatalError et WarningError.
com/example/programmingas3/errors/FatalError.as	Une classe qui définit une erreur FatalError qui peut être renvoyée par l'application. Cette classe étend la classe ApplicationError personnalisée.
com/example/programmingas3/errors/Validator.as	Une classe qui définit une seule méthode qui valide un paquet XML employee fourni par l'utilisateur.
com/example/programmingas3/errors/WarningError.as	Une classe qui définit une erreur WarningError qui peut être renvoyée par l'application. Cette classe étend la classe ApplicationError personnalisée.

Présentation de l'application CustomErrors

Au chargement de l'application, la méthode `initApp()` est appelée dans Flex ou le code du scénario (autre qu'une fonction) est exécuté dans Flash. Ce code définit un exemple de paquet XML que la classe Validator vérifiera. Le code suivant est exécuté :

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

Le paquet XML est ensuite affiché dans une occurrence du composant TextArea sur la scène. Ceci vous permet de modifier le paquet XML avant de tenter de le revalider.

Lorsque l'utilisateur clique sur le bouton de validation, la méthode `validateData()` est appelée. Cette méthode valide le paquet XML employee à l'aide de la méthode `validateEmployeeXML()` de la classe Validator. Le code suivant présente la méthode `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);
    }
}
```

Un objet XML temporaire est d'abord créé à l'aide du contenu de l'occurrence du composant TextArea `xmlText`. Ensuite, la méthode `validateEmployeeXML()` de la classe Validator personnalisée (com.example.programmingas3/errors/Validator.as) est appelée et transmet l'objet XML temporaire comme paramètre. Si le paquet XML est valide, l'occurrence du composant Label `status` affiche un message de réussite et l'application se ferme. Si la méthode `validateEmployeeXML()` renvoie une erreur personnalisée (c'est-à-dire qu'une erreur FatalError, WarningError ou une erreur générique se produit), l'instruction `catch` appropriée s'exécute et appelle les méthodes `showFatalError()`, `showWarningError()` ou `showGenericError()`. Chacune de ces méthodes affiche un message approprié dans une zone de texte appelée `statusTex`t pour avertir l'utilisateur de l'erreur spécifique qui s'est produite. Chaque méthode met également à jour l'occurrence du composant Label `status` avec un message spécifique.

Si une erreur grave se produit pendant une tentative de validation du paquet XML employee, le message d'erreur s'affiche dans la zone de texte `statusText` et l'occurrence du composant TextArea `xmlText` et l'occurrence du composant Button `validateBtn` sont désactivées, comme l'indique le code suivant :

```
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();
}
```

S'il se produit une erreur d'avertissement au lieu d'une erreur grave, le message d'erreur est affiché dans l'occurrence de TextArea `statusText`, mais les occurrences de composant TextField `xmlText` et Button ne sont pas désactivées. La méthode `showWarningError()` affiche le message d'erreur personnalisé dans la zone de texte `statusText`. Le message invite également l'utilisateur à indiquer s'il souhaite poursuivre la validation du code XML ou abandonner le script. Le fragment de code suivant présente la méthode `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;
}
```

Lorsque l'utilisateur clique sur le bouton Oui ou Non, la méthode `closeHandler()` est appelée. Le fragment de code suivant présente la méthode `closeHandler()` :

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

Si l'utilisateur souhaite abandonner le script en cliquant sur Oui, une exception FatalError est renvoyée, provoquant l'arrêt de l'application.

Création d'une validation personnalisée

La classe de validation personnalisée contient une seule méthode, `validateEmployeeXML()`. La méthode `validateEmployeeXML()` prend un seul argument, `employee`, qui correspond au paquet XML que vous souhaitez valider. La méthode `validateEmployeeXML()` est la suivante :

```
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);
    }
}
```

Un employé doit appartenir à un (et un seul) centre de coût pour être validé. S'il n'appartient à aucun centre de coût, la méthode renvoie une erreur FatalError, qui se propage jusqu'à la méthode `validateData()` dans le fichier d'application principale. Si l'employé appartient à plusieurs centres de coût, une erreur WarningError est renvoyée. La dernière vérification de la validation XML contrôle que l'utilisateur possède un seul numéro de sécurité sociale défini (le nœud `ssn` dans le paquet XML). S'il n'y a pas exactement un nœud `ssn`, une erreur FatalError est renvoyée.

Vous pouvez ajouter des vérifications supplémentaires à la méthode `validateEmployeeXML()`. Par exemple, pour vérifier que le nœud `ssn` contient un numéro valide, ou que l'employé possède au moins un numéro de téléphone et une adresse électronique définis, et que ces deux valeurs sont valides. Vous pouvez également modifier le XML de façon à ce que chaque employé ait un ID unique et spécifie l'ID de son responsable.

Définition de la classe ApplicationError

La classe ApplicationError sert de classe de base aux classes FatalError et WarningError. Elle étend la classe Error et définit ses propres propriétés et méthodes personnalisées, y compris un ID d'erreur, la gravité et un objet XML contenant les codes d'erreur personnalisés et les messages. Cette classe définit également deux constantes statiques utilisées pour définir la gravité de chaque type d'erreur.

La méthode du constructeur de la classe ApplicationError est la suivante :

```
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>;
}
```

Chaque nœud d'erreur dans l'objet XML contient un code numérique unique et un message d'erreur. Vous pouvez consulter facilement les messages d'erreur par code d'erreur à l'aide d'E4X, comme indiqué dans la méthode `getMessageText()` suivante :

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

La méthode `getMessageText()` prend un seul argument entier, `id`, et renvoie une chaîne. L'argument `id` correspond au code de l'erreur à rechercher. Par exemple, lorsque vous transmettez un `id` de 9001, l'erreur indiquant que les employés doivent être affectés à un seul centre de coût est renvoyée. Si plusieurs erreurs ont le même code, ActionScript renvoie le message d'erreur uniquement pour le premier résultat trouvé (`message[0]` dans l'objet XMLList renvoyé).

La méthode suivante de cette classe, `getTitle()`, ne prend aucun paramètre et renvoie une valeur de chaîne qui contient l'ID de cette erreur spécifique. Cette valeur permet d'identifier aisément l'erreur exacte qui s'est produite lors de la validation du paquet XML. Le fragment de code suivant présente la méthode `getTitle()` :

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

La dernière méthode finale de la classe ApplicationError est `toString()`. Cette méthode remplace la fonction définie dans la classe Error pour que vous puissiez personnaliser la présentation du message d'erreur. La méthode renvoie une chaîne qui identifie le numéro d'erreur spécifique et le message qui s'est affiché.

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

Définition de la classe FatalError

La classe FatalError étend la classe ApplicationError personnalisée et définit trois méthodes : le constructeur FatalError, `getTitle()` et `toString()`. La première méthode, le constructeur FatalError, prend un seul argument entier, `errorID`, et définit la gravité de l'erreur à l'aide des valeurs de constante statiques définies dans la classe ApplicationError. Elle obtient le message de l'erreur spécifique en appelant la méthode `getMessageText()` dans la classe ApplicationError. Le constructeur FatalError se présente comme suit :

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

La méthode suivante de la classe FatalError, `getTitle()`, remplace la méthode `getTitle()` définie précédemment dans la classe ApplicationError et ajoute le texte “-- FATAL” dans le titre pour informer l'utilisateur qu'une erreur grave s'est produite. La méthode `getTitle()` se présente comme suit :

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

La méthode finale dans cette classe, `toString()`, remplace la méthode `toString()` définie dans la classe ApplicationError. La méthode `toString()` est la suivante :

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

Définition de la classe WarningError

La classe WarningError étend la classe ApplicationError et est presque identique à la classe FatalError, à l'exception de quelques changements de chaîne mineurs. Elle définit la gravité de l'erreur sur ApplicationError.WARNING au lieu de ApplicationError.FATAL, comme indiqué dans le code suivant :

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

Chapitre 10 : Utilisation d'expressions régulières

Une expression régulière décrit un modèle servant à rechercher et à manipuler du texte de correspondance dans des chaînes. Les expressions régulières ressemblent à des chaînes, mais elles comprennent des codes spéciaux pour décrire des modèles et des répétitions. Par exemple, l'expression régulière suivante correspond à une chaîne qui commence par le caractère A suivi d'un ou de plusieurs chiffres séquentiels :

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

Ce chapitre décrit la syntaxe de base permettant de construire des expressions régulières. Néanmoins, les expressions régulières peuvent être très complexes et comporter de nombreuses nuances. Vous pouvez vous documenter sur les expressions régulières sur le Web et dans les librairies. Différents environnements de programmation implémentent des expressions régulières de différentes façons. ActionScript 3.0 implémente des expressions régulières comme défini dans la version 3 de la spécification du langage ECMAScript (ECMA-262).

Principes de base des expressions régulières

Introduction à l'utilisation d'expressions régulières

Une expression régulière décrit un modèle de caractères. Les expressions régulières servent généralement à vérifier qu'une valeur de texte est conforme à un modèle particulier (par exemple, vérifier qu'un numéro de téléphone saisi par l'utilisateur comporte le nombre de chiffres correct) ou à remplacer des portions d'une valeur de texte qui correspondent à un modèle donné.

Les expressions régulières peuvent être simples. Par exemple, supposons que vous souhaitiez confirmer qu'une chaîne particulière correspond à ABC ou que vous souhaitiez remplacer chaque occurrence d'ABC dans une chaîne par un autre texte. Dans ce cas, vous pouvez utiliser l'expression régulière suivante qui définit le modèle comportant les lettres A, B et C, dans l'ordre :

```
/ABC/
```

Le littéral de l'expression régulière est délimité avec la barre oblique (`/`).

Les modèles d'expression régulière peuvent également être complexes et parfois sembler obscures, comme l'expression suivante pour établir une correspondance avec une adresse électronique valide :

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

Vous utiliserez le plus souvent des expressions régulières pour rechercher des modèles dans des chaînes et pour remplacer des caractères. Dans ces cas, vous créerez un objet d'expression régulière et l'utiliserez comme paramètre pour une ou plusieurs méthodes de classe String. Les méthodes suivantes de la classe String prennent des expressions régulières comme paramètres : `match()`, `replace()`, `search()` et `split()`. Pour plus d'informations sur ces méthodes, consultez la section « Recherche de modèles dans des chaînes et remplacement de sous-chaînes » à la page 150.

La classe RegExp comprend les méthodes suivantes : `test()` et `exec()`. Pour plus d'informations, consultez la section « Méthodes d'utilisation d'expressions régulières avec des chaînes » à la page 226.

Tâches d'expression régulière courantes

Les différentes utilisations courantes d'expressions régulières sont décrites en détail dans ce chapitre :

- Création d'un modèle d'expression régulière
- Utilisation de caractères spéciaux dans des modèles
- Identification de séquences de plusieurs caractères (un nombre à deux chiffres ou entre sept et dix lettres, par exemple)
- Identification d'un caractère dans une série de lettres ou de nombres (toute lettre de *a* à *m*)
- Identification d'un caractère dans un jeu de caractères possibles
- Identification de sous-séquences (segments dans un modèle)
- Mise en correspondance et substitution de texte selon des modèles

Concepts importants et terminologie

La liste de référence suivante contient les termes importants utilisés dans ce chapitre :

- Caractère d'échappement : un caractère indiquant que le caractère qui suit doit être considéré comme un caractère de remplacement plutôt que comme un caractère littéral. Dans une syntaxe d'expression régulière, la barre oblique inverse (`\`) est le caractère d'échappement. Par conséquent, une barre oblique inverse suivie d'un autre caractère est un code spécial plutôt que simplement le caractère lui-même.
- Indicateur : un caractère qui spécifie une option concernant la façon dont le modèle d'expression régulière doit être utilisé (s'il faut respecter la casse, par exemple).
- Caractère de remplacement : un caractère qui a une signification spéciale dans un modèle d'expression régulière et qui ne représente pas littéralement ce caractère dans le modèle.
- Quantificateur : un caractère (ou plusieurs caractères) indiquant le nombre de fois qu'un partie du modèle doit se répéter. Par exemple, un quantificateur peut être utilisé pour indiquer qu'un code postal américain doit contenir cinq ou neuf chiffres.
- Expression régulière : une instruction de programme définissant un modèle de caractères pouvant être utilisé pour confirmer si d'autres chaînes correspondent à ce modèle ou pour remplacer des portions d'une chaîne.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester des exemples de code. Etant donné que les codes de ce chapitre sont constitués principalement de modèles d'expression régulière, le test des exemples implique que vous suiviez les étapes suivantes :

1. Créez un nouveau document Flash.
2. Sélectionnez une image-clé et ouvrez le panneau Actions.
3. Créez une variable RegExp (expression régulière) comme la suivante :

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

4. Copiez le modèle de l'exemple et affectez-le comme valeur de votre variable RegExp. Par exemple, dans la ligne de code précédente, le modèle fait partie du code situé à droite du signe d'égalité, point-virgule exclu (`/ABC/`).
5. Créez une ou plusieurs variables contenant des chaînes appropriées au test de votre expression régulière. Par exemple, si vous créez une expression régulière pour tester des adresses électroniques valides, créez quelques variables String contenant des adresses électroniques valides et non valides :

```
var goodEmail:String = "bob@example.com";
var badEmail:String = "5@$2.99";
```

6 Ajoutez des lignes de code pour tester les variables String afin de déterminer si elles correspondent au modèle d'expression régulière. Celles-ci seront les valeurs que vous afficherez à l'écran à l'aide de la fonction `trace()` ou en les écrivant dans un champ de texte sur la scène.

```
trace(goodEmail, " is valid:", pattern.test(goodEmail));
trace(badEmail, " is valid:", pattern.test(badEmail));
```

Par exemple, supposons que `pattern` définisse le modèle d'expression régulière pour une adresse électronique valide. Les lignes de code précédentes écrivent ce texte dans le panneau Sortie:

```
bob@example.com is valid: true
5@$2.99 is valid: false
```

Pour plus d'informations sur le test des valeurs en les écrivant dans une occurrence de champ de texte sur la Scène ou en utilisant la fonction `trace()` pour imprimer les valeurs dans le panneau Sortie, consultez la section « Test des exemples de code contenus dans un chapitre » à la page 36.

Syntaxe d'expression régulière

Cette section décrit tous les éléments de la syntaxe d'expression régulière d'ActionScript. Comme vous pourrez le constater, les expressions régulières peuvent être très complexes et comporter de nombreuses nuances. Vous pouvez vous documenter sur les expressions régulières sur le Web et dans les librairies. Différents environnements de programmation implémentent des expressions régulières de différentes façons. ActionScript 3.0 implémente des expressions régulières comme défini dans la version 3 de la spécification du langage ECMAScript (ECMA-262).

Généralement, vous utilisez des expressions régulières qui correspondent à des modèles plus compliqués qu'une simple chaîne de caractères. Par exemple, l'expression régulière suivante définit le modèle comportant les lettres A, B et C, dans l'ordre, suivies par un chiffre :

```
/ABC\d/
```

Le code `\d` représente un chiffre. La barre oblique inverse (`\`) est appelée caractère d'échappement. Lorsqu'elle est combinée au caractère qui la suit (dans ce cas, la lettre d), elle a une signification spéciale dans l'expression régulière. Ce chapitre décrit ces séquences de caractères d'échappement et d'autres fonctions de syntaxe d'expression régulière.

L'expression régulière suivante définit le modèle des lettres ABC suivies par des chiffres (remarquez l'astérisque) :

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

L'astérisque (`*`) est un *caractère de remplacement*. Un caractère de remplacement est un caractère ayant une signification spéciale dans les expressions régulières. L'astérisque est un type de caractère de remplacement spécifique appelé *quantificateur*, utilisé pour quantifier le nombre de répétitions d'un caractère ou groupe de caractères. Pour plus de détails, consultez la section « Quantificateurs » à la page 218.

Outre son modèle, une expression régulière peut contenir des indicateurs qui spécifient comment l'expression régulière doit être mise en correspondance. Par exemple, l'expression régulière suivante utilise l'indicateur `i` qui indique qu'elle ignore le respect de la casse dans les chaînes de correspondance :

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

Pour plus d'informations, consultez la section « Indicateurs et propriétés » à la page 223.

Vous pouvez utiliser des expressions régulières à l'aide des méthodes suivantes de la classe String : `match()`, `replace()` et `search()`. Pour plus d'informations sur ces méthodes, consultez la section « Recherche de modèles dans des chaînes et remplacement de sous-chaînes » à la page 150.

Création d'une occurrence d'expression régulière

Il existe deux façons de créer une occurrence d'expression régulière. L'une des méthodes consiste à utiliser des barres obliques (`/`) pour délimiter l'expression régulière, et l'autre consiste à utiliser le constructeur `new`. Par exemple, les expressions régulières suivantes sont équivalentes :

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

Des barres obliques délimitent un littéral d'expression régulière de la même façon que des guillemets délimitent un littéral de chaîne. La partie de l'expression régulière contenue entre les barres obliques définit le *modèle.* L'expression régulière peut également inclure des *indicateurs* après la barre de délimitation finale. Ces indicateurs sont considérés comme faisant partie de l'expression régulière, mais ils sont séparés de son modèle.

Lorsque vous utilisez le constructeur `new`, vous utilisez deux chaînes pour définir l'expression régulière. La première chaîne définit le modèle, et la seconde les indicateurs, comme dans l'exemple suivant :

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

Lorsque vous incluez une barre oblique *dans* une expression régulière qui est définie à l'aide de délimiteurs de barre oblique, vous devez faire précéder la barre oblique du caractère d'échappement (`\`). Par exemple, l'expression régulière suivante correspond au modèle `1/2` :

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

Pour inclure des guillemets *dans* une expression régulière définie avec le constructeur `new`, vous devez ajouter un caractère d'échappement (`\`) avant les guillemets (comme lorsque vous définissez un littéral String). Par exemple, les expressions régulières suivantes correspondent au modèle `eat at "joe's"` :

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

N'utilisez pas le caractère d'échappement avec des guillemets dans des expressions régulières définies à l'aide des délimiteurs de barre oblique. De même, n'utilisez pas le caractère d'échappement avec des barres obliques dans des expressions régulières définies avec le constructeur `new`. Les expressions régulières suivantes sont équivalentes. Elles définissent le modèle `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"', '');
```

De même, dans une expression régulière définie à l'aide du constructeur `new`, tapez deux fois le caractère barre oblique inverse pour utiliser une métaséquence débutant par le caractère barre oblique inverse(`\`), telle que `\d` (qui correspond à n'importe quel chiffre).

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

Vous devez taper deux fois le caractère barre oblique inverse, car le premier paramètre de la méthode constructeur `RegExp()` est une chaîne. Dans un littéral de chaîne, ce caractère doit être entré deux fois pour être interprété comme une barre oblique inverse unique.

La section qui suit décrit la syntaxe servant à définir des modèles d'expression régulière.

Pour plus d'informations, consultez la section « Indicateurs et propriétés » à la page 223.

Caractères, caractères de remplacement et métaséquences

L'expression régulière la plus simple est celle qui correspond à une séquence de caractères, comme dans l'exemple suivant :

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

Néanmoins, les caractères suivants, appelés caractères de remplacement, ont des significations spéciales dans des expressions régulières :

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

Par exemple, l'expression régulière suivante correspond à la lettre A suivie par zéro ou plusieurs occurrences de la lettre B (le caractère de remplacement astérisque indique cette répétition), suivie par la lettre C :

```
/AB*C/
```

Pour inclure un caractère de remplacement sans sa signification spéciale dans un modèle d'expression régulière, vous devez utiliser le caractère d'échappement (`\`). Par exemple, l'expression régulière suivante correspond à la lettre A suivie par la lettre B, suivie par un astérisque, suivie par la lettre C :

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

Une *métaséquence,* comme un caractère de remplacement, a une signification spéciale dans une expression régulière. Une métaséquence est constituée de plusieurs caractères. Les sections suivantes fournissent des détails sur l'utilisation des caractères de remplacement et des métaséquences.

A propos des caractères de remplacement

Le tableau suivant répertorie les caractères de remplacement que vous pouvez utiliser dans des expressions régulières :

Caractère de remplacement	Description
^ (caret)	Etablit une correspondance au début d'une chaîne. Lorsque l'indicateur `m` (`multiline`) est défini, le caret correspond au début d'une ligne également (consultez la section « Indicateurs et propriétés » à la page 223). Lorsqu'il est utilisé au début d'une classe de caractère, le caret indique la négation et non le début d'une chaîne. Pour plus d'informations, consultez la section « Classes de caractère » à la page 217.
$(signe dollar)	Etablit une correspondance à la fin d'une chaîne. Lorsque l'indicateur `m` (`multiline`) est défini, `$` correspond à la position avant une nouvelle ligne (`\n`) également. Pour plus d'informations, consultez la section « Indicateurs et propriétés » à la page 223.
\ (caractère d'échappement)	Echappe la signification spéciale du caractère de remplacement des caractères spéciaux. Vous pouvez également utiliser le caractère d'échappement si vous souhaitez utiliser une barre oblique dans un littéral d'expression régulière, comme dans `/1\/2/` (pour correspondre au caractère 1, suivi par la barre oblique, suivi par le caractère 2).
. (point)	Correspond à un seul caractère. Un point correspond à un caractère de nouvelle ligne (`\n`) uniquement si l'indicateur `s` (`dotall`) est défini. Pour plus d'informations, consultez la section « Indicateurs et propriétés » à la page 223.
* (étoile)	Correspond à l'élément précédent répété zéro ou plusieurs fois. Pour plus de détails, consultez la section « Quantificateurs » à la page 218.
+ (plus)	Correspond à l'élément précédent répété une ou plusieurs fois. Pour plus de détails, consultez la section « Quantificateurs » à la page 218.
? (point d'interrogation)	Correspond à l'élément précédent répété zéro ou une fois. Pour plus de détails, consultez la section « Quantificateurs » à la page 218.

Caractère de remplacement	Description
(et)	Définit des groupes dans l'expression régulière. Utilisez des groupes pour : • confiner le domaine du permutateur | : `/(a|b|c)d/` • définir le domaine d'un quantificateur :`/(walla.){1,2}/` • dans des backreferences. Par exemple, le `\1` dans l'expression régulière suivante correspond à toute correspondance au premier groupe entre parenthèses du modèle : • `/(\w*) est répété : \1/` Pour plus d'informations, consultez la section « Groupes » à la page 220.
[et]	Spécifie une classe de caractère qui définit des correspondances possibles pour un seul caractère : `/[aeiou]/` correspond à l'un des caractères spécifiés. Dans les classes de caractère, utilisez le trait d'union (-) pour désigner une plage de caractères : `/[A-Z0-9]/` correspond aux lettres majuscules A à Z ou aux chiffres 0 à 9. Dans les classes de caractère, insérez un caractère d'échappement pour échapper les caractères] et les caractères -: `/[+\-]\d+/` correspond à + ou à - avant un ou plusieurs chiffres. Dans les classes de caractère, d'autres caractères (qui sont normalement des caractères de remplacement) sont considérés comme des caractères normaux (et non comme des caractères de remplacement), sans qu'une barre oblique inverse soit nécessaire : `/[$]/£` correspond à `$ou à £`. Pour plus d'informations, consultez la section « Classes de caractère » à la page 217.
| *(opérateur de transfert de données)*	Utilisé pour la permutation, pour correspondre à la partie de gauche ou à celle de droite : `/abc|xyz/` correspond à `abc` ou à `xyz`.

A propos des métaséquences

Les métaséquences sont des séquences de caractères ayant une signification spéciale dans un modèle d'expression régulière. Le tableau suivant décrit ces métaséquences :

Métaséquence	Description
{*n*} {*n*,} et {*n*,*n*}	Indique un quantificateur numérique ou une plage de quantificateurs pour l'élément précédent : `/A{27}/` correspond au caractère `A` répété `27` fois. `/A{3}/` correspond au caractère `A` répété `3` fois ou plus. `/A{3,5}/` correspond au caractère `A` répété `3` à `5` fois. Pour plus de détails, consultez la section « Quantificateurs » à la page 218.
`\b`	Etablit une correspondance à la position entre un caractère mot et un caractère non-mot. Si le premier ou le dernier caractère dans la chaîne est un caractère mot, correspond également au début ou à la fin de la chaîne.
`\B`	Etablit une correspondance à la position entre deux caractères mot. Correspond également à la position entre deux caractères non-mot.
`\d`	Correspond à une décimale.
`\D`	Correspond à tout caractère autre qu'un chiffre.
`\f`	Correspond à un caractère de changement de page.

Métaséquence	Description
`\n`	Correspond au caractère de nouvelle ligne.
`\r`	Correspond au caractère de retour de chariot.
`\s`	Correspond à tout caractère d'espace blanc (un espace, une tabulation, une nouvelle ligne ou un caractère de retour de chariot).
`\S`	Correspond à tout caractère autre qu'un caractère d'espace blanc.
`\t`	Correspond au caractère de tabulation.
`\u`***nnnn***	Correspond au caractère Unicode avec le code de caractère spécifié par le nombre hexadécimal *nnnn*. Par exemple, `\u263a` est le caractère smiley.
`\v`	Correspond à un caractère d'avancement vertical.
`\w`	Correspond à un caractère mot (`AZ`–, `az`–, `0-9` ou `_`). `\w` ne correspond pas aux caractères qui ne sont pas anglais tels que `é`, `ñ`, ou `ç`.
`\W`	Correspond à tout caractère autre qu'un caractère mot.
`\\x`***nn***	Correspond au caractère avec la valeur ASCII spécifiée, comme défini par le nombre hexadécimal *nn*.

Classes de caractère

Vous utilisez des classes de caractère pour spécifier une liste de caractères devant correspondre à une position dans l'expression régulière. Vous définissez des classes de caractère avec des crochets (`[` et `]`). Par exemple, l'expression régulière suivante définit une classe de caractère qui remplace `bag`, `beg`, `big`, `bog` ou `bug` :

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

Séquences d'échappement dans des classes de caractère

La plupart des caractères de remplacement et des métaséquences ayant normalement des significations spéciales dans une expression régulière *n'ont pas* ces mêmes significations dans une classe de caractère. Par exemple, dans une expression régulière, l'astérisque est utilisé pour la répétition, mais ce n'est pas le cas lorsqu'il apparaît dans une classe de caractère. La classe de caractère suivante correspond à l'astérisque de façon littérale, avec tout autre caractère répertorié :

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

Cependant, les trois caractères répertoriés dans le tableau suivant fonctionnent comme des caractères de remplacement, avec une signification spéciale, dans des classes de caractère :

Caractère de remplacement	Signification dans des classes de caractère
`]`	Définit la fin de la classe de caractère.
`-`	Définit une plage de caractères (consultez la section suivante, " Plages de caractères dans des classes de caractère ").
`\`	Définit des métaséquences et annule la signification spéciale des caractères de remplacement.

Pour que ces caractères soient reconnus comme caractères littéraux (dans la signification du caractère de remplacement spéciale), vous devez faire précéder le caractère du caractère d'échappement. Par exemple, l'expression régulière suivante inclut une classe de caractère qui correspond à l'un des quatre symboles (`$`, `\`, `]` ou `-`) :

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

Outre les caractères de remplacement qui conservent leurs significations spéciales, les métaséquences suivantes fonctionnent comme des métaséquences dans des classes de caractère :

Métaséquence	Signification dans des classes de caractère
`\n`	Correspond à un caractère de nouvelle ligne.
`\r`	Correspond à un caractère de retour de chariot.
`\t`	Correspond à un caractère de tabulation.
`\u`***nnnn***	Correspond au caractère avec la valeur de point de code Unicode spécifiée (comme défini par le nombre hexadécimal *nnnn*).
`\\x`***nn***	Correspond au caractère avec la valeur ASCII spécifiée (comme défini par le nombre hexadécimal *nn*).

D'autres caractères de remplacement et métaséquences d'expression régulière sont considérés comme des caractères normaux dans une classe de caractère.

Plages de caractères dans des classes de caractère

Utilisez le trait d'union pour spécifier une plage de caractères telle que `A-Z`, `a-z`, ou `0-9`. Ces caractères doivent constituer une plage valide dans le jeu de caractères. Par exemple, la classe de caractère suivante correspond à un caractère compris dans la plage `a-z` ou un chiffre :

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

Vous pouvez également utiliser le code de caractère ASCII `\\x`*nn* pour spécifier une plage par valeur ASCII. Par exemple, la classe de caractère suivante correspond à un caractère d'un jeu de caractères ASCII étendus (é et ê, par exemple) :

```
\\x
```

Classes de caractère niées

Lorsque vous utilisez un caret (`^`) au début d'une classe de caractère, il la nie (tout caractère non répertorié est considéré comme une correspondance). La classe de caractère suivante correspond à tout caractère *sauf* une lettre minuscule (`az–`) ou un chiffre :

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

Vous devez taper le caret (`^`) au *début* d'une classe de caractère pour indiquer la négation. Autrement, vous ajoutez simplement le caret aux caractères dans la classe de caractère. Par exemple, la classe de caractère suivante correspond à un symbole (le caret, notamment) :

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

Quantificateurs

Vous utilisez des quantificateurs pour spécifier des répétitions de caractères ou de séquences dans des modèles, comme suit :

Caractère de remplacement de quantificateur	Description
`*` (étoile)	Correspond à l'élément précédent répété zéro ou plusieurs fois.

Caractère de remplacement de quantificateur	Description
+ (plus)	Correspond à l'élément précédent répété une ou plusieurs fois.
? (point d'interrogation)	Correspond à l'élément précédent répété zéro ou une fois.
{*n*} {*n*,} et {*n*,*n*}	Indique un quantificateur numérique ou une plage de quantificateurs pour l'élément précédent : /A{27}/ correspond au caractère A répété 27 fois. /A{3}/ correspond au caractère A répété 3 fois ou plus. /A{3,5}/ correspond au caractère A répété 3 à 5 fois.

Vous pouvez appliquer un quantificateur à un seul caractère, à une classe de caractère ou à un groupe :

- /a+/ correspond au caractère a répété une ou plusieurs fois.
- /\d+/ correspond à un ou plusieurs chiffres.
- /[abc]+/ correspond à une répétition d'un ou de plusieurs caractères, a, b, ou c.
- /(very,)*/ correspond au mot very suivi par une virgule et un espace répété zéro ou plusieurs fois.

Vous pouvez utiliser des quantificateurs dans des groupes de parenthèses auxquels sont appliqués des quantificateurs. Par exemple, le quantificateur suivant correspond à des chaînes du type word et word-word-word :

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

Par défaut, les expressions régulières effectuent un *greedy matching*. Tout sous-modèle dans l'expression régulière (.*, par exemple) tente de mettre en correspondance autant de caractères que possible dans la chaîne avant de passer à la partie suivante de l'expression régulière. Par exemple, considérez l'expression régulière et la chaîne suivantes :

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

L'expression régulière correspond à la chaîne entière :

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

Supposez, néanmoins, que vous souhaitez établir une correspondance avec un seul groupe ... . Vous pouvez procéder comme suit :

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

Ajoutez un point d'interrogation (?) après les quantificateurs pour qu'ils deviennent des *quantificateurs paresseux*. Par exemple, l'expression régulière suivante, qui utilise le quantificateur paresseux *? , correspond à suivi du nombre minimum de caractères possible (paresseux), suivi de :

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

Lisez attentivement les points suivants concernant les quantificateurs :

- Les quantificateurs {0} et {0,0} n'excluent pas un élément d'une correspondance.
- Ne combinez pas plusieurs quantificateurs, comme dans /abc+*/.
- Le caractère point (.) ne divise pas les lignes en deux à moins de définir l'indicateur s (dotall), même s'il est suivi d'un quantificateur *. Considérons par exemple le code qui suit :

```
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>
```

Pour plus d'informations, consultez la section « Indicateurs et propriétés » à la page 223.

Permutation

Utilisez le caractère `|` (barre) dans une expression régulière pour que son moteur tienne compte des autres possibilités pour une correspondance. Par exemple, l'expression régulière suivante correspond à l'un des mots `cat, dog, pig, rat`:

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

Vous pouvez utiliser des parenthèses pour définir des groupes et limiter le domaine du permutateur |. L'expression régulière suivante correspond à `cat` suivi de `nap` ou `nip` :

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

Pour plus d'informations, consultez la section « Groupes » à la page 220.

Les deux expressions régulières suivantes (l'une utilisant le permutateur `|`, l'autre une classe de caractère (définie avec `[` et `]`)) sont équivalentes :

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

Pour plus d'informations, consultez la section « Classes de caractère » à la page 217.

Groupes

Vous pouvez spécifier un groupe dans une expression régulière en utilisant des parenthèses, comme suit :

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

Un groupe est une sous-section d'un modèle. Vous pouvez utiliser des groupes pour effectuer les opérations suivantes :

- Appliquer un quantificateur à plusieurs caractères
- Délimiter des sous-modèles à appliquer avec la permutation (à l'aide du caractère `|`)
- Capturer des correspondances de sous-chaîne (par exemple, en utilisant `\1` dans une expression régulière pour établir une correspondance avec un groupe mis en correspondance précédemment, ou en utilisant `$1` de la même façon dans la méthode `replace()` de la classe String)

Les sections suivantes fournissent des détails sur ces utilisations de groupes.

Utilisation de groupes avec des quantificateurs

Si vous n'utilisez pas de groupe, un quantificateur s'applique au caractère ou à la classe de caractère qui le précède, comme indiqué ci-dessous :

```
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
```

Néanmoins, vous pouvez utiliser un groupe pour appliquer un quantificateur à plusieurs caractères ou classes de caractère :

```
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
```

Pour plus d'informations sur les quantificateurs, consultez la section « Quantificateurs » à la page 218.

Utilisation de groupes avec le permutateur (|)

Vous pouvez utiliser des groupes pour définir le groupe de caractères auquel vous souhaitez appliquer un permutateur (|), comme suit :

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

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

Utilisation de groupes pour capturer des correspondances de sous-chaîne

Lorsque vous définissez un groupe entre parenthèses standard dans un modèle, vous pouvez ensuite vous y référer dans l'expression régulière. Il s'agit d'une *backreference*. Ces types de groupes sont appelés *groupes capturés*. Par exemple, dans l'expression régulière suivante, la séquence `\1` correspond à toute sous-chaîne mise en correspondance avec le groupe entre parenthèses capturé :

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

Vous pouvez spécifier jusqu'à 99 backreferences dans une expression régulière en tapant `\1`,`\2`,`...`,`\99`.

De même, dans la méthode `replace()` de la classe String, vous pouvez utiliser `$1-$99` pour insérer des correspondances de chaîne de groupe capturé dans la chaîne de remplacement :

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

Si vous utilisez des groupes capturés, la méthode `exec()` de la classe RegExp et la méthode `match()` de la classe String renvoient des sous-chaînes qui correspondent aux groupes capturés :

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

Utilisation de groupes non capturés et de groupes d'anticipation

Un groupe non capturé est utilisé pour le regroupement uniquement ; il n'est pas collecté et il ne correspond pas à des backreferences numérotées. Utilisez `(?:` et `)` pour définir des groupes non capturés, comme suit :

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

Par exemple, notez la différence entre mettre `(com|org)` dans un groupe capturé et dans un groupe non capturé (la méthode `exec()` répertorie les groupes capturés après la correspondance complète) :

```
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
```

Un type spécial de groupe non capturé est le *groupe d'animation* dont il existe deux types : le *groupe d'animation positif* et le *groupe d'animation négatif.*

Utilisez `(?=` et `)` pour définir un groupe d'anticipation positif, qui spécifie que le sous-modèle dans le groupe doit établir une correspondance à la position. Néanmoins, la portion de la chaîne qui correspond au groupe d'anticipation positif peut correspondre aux modèles restants dans l'expression régulière. Par exemple, étant donné que `(?=e)` est un groupe d'anticipation positif dans le code suivant, le caractère `e` auquel il correspond peut être mis en correspondance par une partie suivante de l'expression régulière (dans ce cas, le groupe capturé, `\w*)` :

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

Utilisez `(?=` et `)` pour définir un groupe d'anticipation négatif, qui indique que le sous-modèle dans le groupe ne doit *pas* établir une correspondance à la position. Par exemple :

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

Utilisation de groupes nommés

Un groupe nommé est un type de groupe dans une expression régulière ayant un identificateur nommé. Utilisez `(?P<name>` et `)` pour définir le groupe nommé. Par exemple, l'expression régulière suivante inclut un groupe nommé avec l'identificateur nommé `digits` :

```
var pattern = /[a-z]+(?P<digits>\d+)[a-z]+/;
```

Lorsque vous utilisez la méthode `exec()`, un groupe nommé de correspondance est ajouté comme propriété du tableau `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
```

Voici un autre exemple, qui utilise deux groupes nommés, avec les identificateurs `name` et `dom` :

```
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
```

Remarque : *les groupes nommés ne font pas partie de la spécification du langage ECMAScript. Ils représentent une fonction ajoutée dans ActionScript 3.0.*

Indicateurs et propriétés

Le tableau suivant répertorie les cinq indicateurs que vous pouvez définir pour des expressions régulières. Vous pouvez accéder à chaque indicateur en tant que propriété de l'objet d'expression régulière.

Indicateur	Propriété	Description
`g`	`global`	A plusieurs correspondances.
`i`	`ignoreCase`	Correspondance sensible à la casse. S'applique aux caractères `A`—`Z` et `a`—`z` mais pas à des caractères étendus tels que `É` et `é` .
`m`	`multiline`	Lorsque cet indicateur est défini, `$` et `^` peuvent correspondre au début d'une ligne et à la fin d'une ligne, respectivement.
`s`	`dotall`	Lorsque cet indicateur est défini, `.` (point) peut correspondre au caractère de nouvelle ligne (`\n`).
`x`	`extended`	Autorise les expressions régulières étendues. Vous pouvez tapez des espaces dans l'expression régulière. Ils sont ignorés dans le cadre du modèle. Ceci vous permet de taper un code d'expression régulière de façon plus lisible.

Ces propriétés sont en lecture seule. Vous pouvez définir les indicateurs (`g`, `i`, `m`, `s`, `x`) lorsque vous définissez une variable d'expression régulière, comme suit :

```
var re:RegExp = /abc/gimsx;
```

Cependant, vous ne pouvez pas définir directement les propriétés nommées. Par exemple, le code suivant génère une erreur :

```
var re:RegExp = /abc/;
re.global = true; // This generates an error.
```

Par défaut, à moins de les spécifier dans la déclaration d'expression régulière, les indicateurs ne sont pas définis, et les propriétés correspondantes sont également définies sur `false`.

De plus, il existe deux autres propriétés d'une expression régulière :

- La propriété `lastIndex` spécifie la position d'index dans la chaîne à utiliser pour l'appel suivant à la méthode `exec()` ou `test()` d'une expression régulière.
- La propriété `source` spécifie la chaîne qui définit la portion de modèle de l'expression régulière.

L'indicateur g (global)

Lorsque l'indicateur `g` (`global`) n'est *pas* inclus, une expression régulière n'a pas plus d'une correspondance. Par exemple, avec l'indicateur `g` exclu de l'expression régulière, la méthode `String.match()` renvoie une sous-chaîne de correspondance uniquement :

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/;
trace(str.match(pattern)) // output: she
```

Lorsque l'indicateur `g` est défini, la méthode `Sting.match()` renvoie plusieurs correspondances, comme suit :

```
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
```

L'indicateur i (ignoreCase)

Par défaut, les correspondances d'expression régulière sont sensibles à la casse. Lorsque vous définissez l'indicateur `i` (`ignoreCase`), le respect de la casse est ignoré. Par exemple, la lettre minuscule `s` dans l'expression régulière ne correspond pas à la lettre majuscule `S`, le premier caractère de la chaîne :

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/)); // output: 13 -- Not the first character
```

Lorsque l'indicateur `i` est défini, cependant, l'expression régulière correspond à la lettre majuscule `S` :

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/i)); // output: 0
```

L'indicateur `i` ignore le respect de la casse uniquement pour les caractères `A–Z` et `a–z`, mais par pour les caractères étendus tels que `É` et `é` .

L'indicateur m (multiline)

Si l'indicateur `m` (`multiline`) n'est pas défini, le `^` correspond au début de la chaîne et le `$` à sa fin. Si l'indicateur `m` est défini, ces caractères correspondent au début d'une ligne et à la fin d'une ligne, respectivement. Considérez la chaîne suivante, qui inclut un caractère de nouvelle ligne :

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/g)); // Match a word at the beginning of the string.
```

Même si l'indicateur `g` (`global`) est défini dans l'expression régulière, la méthode `match()` correspond à une seule sous-chaîne, car il n'existe qu'une seule correspondance pour le `^` (le début de la chaîne). Le résultat est le suivant :

```
Test
```

Voici le même code avec l'indicateur `m` défini :

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/gm)); // Match a word at the beginning of lines.
```

Cette fois, le résultat inclut les mots au début des deux lignes :

```
Test,Multiline
```

Seul le caractère `\n` signale la fin d'une ligne. Ce n'est pas le cas des caractères suivants :

- Retour de chariot (`\r`)

- Séparateur de ligne Unicode (`\u2028`)
- Séparateur de paragraphe Unicode (`\u2028`)

L'indicateur s (dotall)

Si l'indicateur `s` (`dotall` ou “dot all”) n'est pas défini, un point (`.`) dans un modèle d'expression régulière ne correspond pas à un caractère de nouvelle ligne (`\n`). Par conséquent, il n'existe aucune correspondance pour l'exemple suivant :

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/;
trace(str.match(re));
```

Néanmoins, si l'indicateur `s` est défini, le point correspond au caractère de nouvelle ligne :

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/s;
trace(str.match(re));
```

Dans ce cas, la correspondance est la sous-chaîne entière dans les balises `<p>`, y compris le caractère de nouvelle ligne :

```
<p>Test
Multiline</p>
```

L'indicateur x (extended)

Les expressions régulières peuvent être difficiles à lire, notamment lorsqu'elles comprennent de nombreux métasymboles et métaséquences. Par exemple :

```
/<p(>|(\s*[^>]*>)).*?<\/p>/gi
```

Lorsque vous utilisez l'indicateur `x` (`extended`) dans une expression régulière, les espaces vides que vous tapez dans le modèle sont ignorés. Par exemple, l'expression régulière suivante est identique à l'exemple précédent :

```
/     <p    (>  |  (\s*  [^>]*  >))    .*?    <\/p>  /gix
```

Si l'indicateur `x` est défini et que vous souhaitez établir une correspondance avec un espace vide, faites précéder l'espace vide d'une barre oblique. Par exemple, les deux expressions régulières suivantes sont équivalentes :

```
/foo bar/
/foo \ bar/x
```

La propriété lastIndex

La propriété `lastIndex` spécifie la position d'index dans la chaîne à laquelle la recherche suivante doit démarrer. Cette propriété affecte les méthodes `exec()` et `test()` appelées sur une expression régulière dont l'indicateur `g` est défini sur `true`. Considérons par exemple le code qui suit :

```
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);
}
```

La propriété `lastIndex` est défini sur 0 par défaut (pour commencer les recherches au début de la chaîne). Après chaque correspondance, elle est définie sur la position d'index suivant la correspondance. Par conséquent, le résultat pour le code précédent est le suivant :

```
0
5
11
18
25
36
44
```

Si l'indicateur `global` est défini sur `false`, les méthodes `exec()` et `test()` n'utilisent pas ni ne définissent la propriété `lastIndex`.

Les méthodes `match()`, `replace()` et `search()` de la classe String lancent toutes les recherches du début de la chaîne, indépendamment du réglage de la propriété `lastIndex` de l'expression régulière utilisée dans l'appel à la méthode. (Néanmoins, la méthode `match()` définit `lastIndex` sur 0.)

Vous pouvez définir la propriété `lastIndex` de sorte à ajuster la position de début dans la chaîne pour la mise en correspondance des expressions régulières.

La propriété source

La propriété `source` spécifie la chaîne qui définit la portion de modèle d'une expression régulière. Par exemple :

```
var pattern:RegExp = /foo/gi;
trace(pattern.source); // foo
```

Méthodes d'utilisation d'expressions régulières avec des chaînes

La classe RegExp comprend deux méthodes : `exec()` et `test()`.

Outre les méthodes `exec()` et `test()` de la classe RegExp, la classe String comprend les méthodes suivantes qui vous permettent d'établir une correspondance avec des expressions régulières dans des chaînes : `match()`, `replace()`, `search()` et `splice()`.

La méthode test()

La méthode `test()` de la classe RegExp vérifie si la chaîne fournie contient une correspondance pour l'expression régulière, comme l'indique l'exemple suivant :

```
var pattern:RegExp = /Class-\w/;
var str = "Class-A";
trace(pattern.test(str)); // output: true
```

La méthode exec()

La méthode `exec()` de la classe RegExp vérifie si la chaîne fournie contient une correspondance pour l'expression régulière et renvoie un tableau avec les éléments suivants :

- La sous-chaîne correspondante
- Les correspondances de sous-chaîne pour les groupes entre parenthèses dans l'expression régulière

Le tableau inclut également une propriété `index` qui indique la position d'index du début de la correspondance de sous-chaîne.

Considérons par exemple le code qui suit :

```
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
```

Utilisez la méthode `exec()` plusieurs fois pour établir une correspondance avec plusieurs sous-chaînes lorsque l'indicateur `g` (`global`) est défini pour l'expression régulière :

```
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
```

Méthodes de chaîne utilisant des paramètres RegExp

Les méthodes suivantes de la classe String prennent des expressions régulières comme paramètres : `match()`, `replace()`, `search()` et `split()`. Pour plus d'informations sur ces méthodes, consultez la section « Recherche de modèles dans des chaînes et remplacement de sous-chaînes » à la page 150.

Exemple : un analyseur Wiki

Cet exemple simple de conversion de texte Wiki illustre des utilisations d'expressions régulières :

- Conversion de lignes de texte qui mettent en correspondance un modèle Wiki source et les chaînes de sortie HTML appropriées.
- Utilisation d'une expression régulière pour convertir des modèles URL en balises de lien hypertexte HTML `<a>`
- Utilisation d'une expression régulière pour convertir les chaînes dollar américain (`"$9.95"`, par exemple) en chaînes euro (`"8.24 €"`, par exemple)

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers de l'application WikiEditor se trouvent dans le dossier Samples/WikiEditor. L'application se compose des fichiers suivants :

Fichier	Description
WikiEditor.mxml ou WikiEditor.fla	Le fichier d'application principal dans Flash (FLA) ou Flex (MXML).
com/example/programmingas3/regExpExamples/WikiParser.as	Une classe qui comprend des méthodes utilisant des expressions régulières pour convertir des modèles de texte d'entrée Wiki en sortie HTML équivalente.
com/example/programmingas3/regExpExamples/URLParser.as	Une classe qui comprend des méthodes utilisant des expressions régulières pour convertir des chaînes URL en balises de lien hypertexte HTML `<a>`.
com/example/programmingas3/regExpExamples/CurrencyConverter.as	Une classe qui comprend des méthodes utilisant des expressions régulières pour convertir des chaînes dollar américain en chaînes euro.

Définition de la classe WikiParser

La classe WikiParser inclut des méthodes qui convertissent le texte d'entrée Wiki en sortie HTML équivalente. Il ne s'agit pas d'une application de conversion Wiki très puissante, mais elle illustre des utilisations d'expressions régulières pour la mise en correspondance de modèle et la conversion de chaîne.

La fonction constructeur et la méthode `setWikiData()` initialisent un exemple de chaîne de texte d'entrée Wiki, comme suit :

```
public function WikiParser()
{
    wikiData = setWikiData();
}
```

Lorsque l'utilisateur clique sur le bouton de test dans l'application exemple, cette dernière appelle la méthode `parseWikiString()` de l'objet WikiParser. Cette méthode appelle plusieurs autres méthodes, qui assemblent à leur tour la chaîne HTML résultante.

```
public function parseWikiString(wikiString:String):String
{
    var result:String = parseBold(wikiString);
    result = parseItalic(result);
    result = linesToParagraphs(result);
    result = parseBullets(result);
    return result;
}
```

Chaque méthode appelée—`parseBold()`, `parseItalic()`, `linesToParagraphs()`, et `parseBullets()`—utilise la méthode `replace()` de la chaîne pour remplacer les modèles de correspondance, définis par une expression régulière, de façon à transformer le texte Wiki en texte formaté HTML.

Conversion des modèles de texte en gras et en italique

La méthode `parseBold()` recherche un modèle de texte Wiki en gras (`'''foo'''`, par exemple) et le transforme en son équivalent HTML (`<b>foo</b>`, par exemple), comme suit :

```
private function parseBold(input:String):String
{
    var pattern:RegExp = /'''(.*?)'''/g;
    return input.replace(pattern, "<b>$1</b>");
}
```

Notez que la portion `(.?*)` de l'expression régulière correspond à des caractères (`*`) entre les deux modèles de définition `'''`. Le quantificateur `?` rend la correspondance nongreedy, de façon à ce que pour une chaîne telle que `'''aaa''' bbb '''ccc'''`, la première chaîne mise en correspondance soit `'''aaa'''` et non la chaîne entière (qui commence et se termine par le modèle `'''`).

Les parenthèses dans l'expression régulière définissent un groupe capturé, et la méthode `replace()` se réfère à ce groupe en utilisant le code `$1` dans la chaîne de remplacement. L'indicateur `g` (`global`) dans l'expression régulière vérifie que la méthode `replace()` remplace toutes les correspondances dans la chaîne (pas simplement la première).

La méthode `parseItalic()` fonctionne comme la méthode `parseBold()`, sauf qu'elle recherche deux apostrophes (`''`) comme séparateur pour le texte en italique (au lieu de trois) :

```
private function parseItalic(input:String):String
{
    var pattern:RegExp = /''(.*?)''/g;
    return input.replace(pattern, "<i>$1</i>");
}
```

Conversion de modèles de puce

Comme dans l'exemple suivant, la méthode `parseBullet()` recherche le modèle de puce Wiki (`* foo`, par exemple) et le transforme en son équivalent HTML (`<li>foo</li>`, par exemple) :

```
private function parseBullets(input:String):String
{
    var pattern:RegExp = /^\*(.*)/gm;
    return input.replace(pattern, "<li>$1</li>");
}
```

Le symbole `^` au début de l'expression régulière correspond au début d'une ligne. L'indicateur `m` (`multiline`) dans l'expression régulière fait en sorte que cette dernière compare le symbole `^` au début d'une ligne, et non simplement au début de la chaîne.

Le modèle `\*` correspond à un astérisque (la barre oblique est utilisée pour signaler un astérisque littéral au lieu d'un quantificateur `*`).

Les parenthèses dans l'expression régulière définissent un groupe capturé, et la méthode `replace()` se réfère à ce groupe en utilisant le code `$1` dans la chaîne de remplacement. L'indicateur `g` (`global`) dans l'expression régulière vérifie que la méthode `replace()` remplace toutes les correspondances dans la chaîne (pas simplement la première).

Conversion de modèles Wiki de paragraphe

La méthode `linesToParagraphs()` convertit chaque ligne dans la chaîne Wiki d'entrée en une balise de paragraphe HTML `<p>`. Ces lignes dans la méthode extraient des lignes vides de la chaîne d'entrée Wiki :

```
var pattern:RegExp = /^$/gm;
var result:String = input.replace(pattern, "");
```

Les symboles `^` et `$` dans l'expression régulière correspondent au début et à la fin d'une ligne. L'indicateur `m` (`multiline`) dans l'expression régulière fait en sorte que cette dernière compare le symbole ^ au début d'une ligne, et non simplement au début de la chaîne.

La méthode `replace()` remplace toutes les chaînes de correspondance (lignes vides) par une chaîne vide (`""`). L'indicateur `g` (`global`) dans l'expression régulière vérifie que la méthode `replace()` remplace toutes les correspondances dans la chaîne (pas simplement la première).

Conversion d'URL en balises HTML <a>

Lorsque l'utilisateur clique sur le bouton de test dans l'exemple d'application et qu'il a coché la case `urlToATag`, l'application appelle la méthode statique `URLParser.urlToATag()` pour convertir les chaînes URL de la chaîne Wiki d'entrée en balises HTML `<a>`.

```
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>");
```

La fonction constructeur `RegExp()` sert à assembler une expression régulière (`urlPattern`) à partir de plusieurs parties constituantes. Ces parties constituantes sont représentées par chaque chaîne définissant une partie du modèle de l'expression régulière.

La première partie du modèle de l'expression régulière, définie par la chaîne `protocol`, définit un protocole URL : `http://` ou `ftp://`. Les parenthèses définissent un groupe non capturé, indiqué par le symbole `?`. Ceci signifie que les parenthèses servent simplement à définir un groupe pour le modèle de permutation `|` ; le groupe ne correspondra pas à des codes de backreference (`$1`, `$2`, `$3`) dans la chaîne de remplacement de la méthode `replace()`.

Les autres parties constituantes de l'expression régulière utilisent chacune des groupes capturés (indiqués par des parenthèses dans le modèle) qui sont ensuite utilisés dans les codes de backreference (`$1`, `$2`, `$3`) dans la chaîne de remplacement de la méthode `replace()`.

La partie du modèle définie par la chaîne `urlPart` correspond *au moins* à l'un des caractères suivants : `a-z`, `0-9`, _ ou `-`. La quantificateur `+` indique qu'au moins un caractère a une correspondance. Le `\.` indique un point obligatoire (`.`). Et le reste correspond à une autre chaîne d'au moins un de ces caractères : `a-z`, `0-9`, _ ou `-`.

La partie du modèle définie par la chaîne `optionalUrlPart` correspond à *zéro ou plus* des caractères suivants : un point (`.`) suivi par n'importe quel nombre de caractères alphanumériques (y compris _ et `-`). Le quantificateur `*` indique que zéro ou plus de caractères ont une correspondance.

L'appel à la méthode `replace()` utilise l'expression régulière et assemble la chaîne HTML de remplacement, à l'aide de backreferences.

La méthode `urlToATag()` appelle ensuite la méthode `emailToATag()`, qui utilise des techniques semblables pour remplacer des modèles de courrier électronique par des chaînes de lien hypertexte HTML `<a>`. Les expressions régulières utilisées pour correspondre à des URL HTTP, FTP et de courrier électronique dans le fichier sont assez simples car elles sont données à titre d'exemple. Elles sont beaucoup plus compliquées si l'on souhaite établir une correspondance plus correcte.

Conversion des chaînes dollar américain en chaînes euro

Lorsque l'utilisateur clique sur le bouton de test dans l'exemple d'application et qu'il a coché la case `dollarToEuro` , l'application appelle la méthode statique `CurrencyConverter.usdToEuro()` pour convertir des chaînes dollar américain (`"$9.95"`, par exemple) en chaînes euro (`"8.24€"`, par exemple), comme suit :

```
var usdPrice:RegExp = /\$([\d,]+.\d+)+/g;
return input.replace(usdPrice, usdStrToEuroStr);
```

La première ligne définit un modèle simple pour établir une correspondance avec des chaînes dollar américain. Le caractère `$` est précédé du caractère d'échappement (`\`).

La méthode `replace()` utilise l'expression régulière pour établir une correspondance avec le modèle et appelle la fonction `usdStrToEuroStr()` pour déterminer la chaîne de remplacement (une valeur en euros).

Lorsqu'un nom de fonction est utilisé comme deuxième paramètre de la méthode `replace()`, les éléments suivants sont transmis comme paramètres à la fonction appelée :

- La partie correspondante de la chaîne.
- Tout groupe entre parenthèses capturé correspondant. Le nombre d'arguments transmis de cette façon varie selon le nombre de correspondances de groupes entre parenthèses capturées. Pour déterminer le nombre de correspondances de groupes entre parenthèses capturés, vérifiez `arguments.length - 3` dans le code de la fonction.
- La position d'index dans la chaîne où débute la correspondance.
- La chaîne complète.

La méthode `usdStrToEuroStr()` convertit les modèles de chaîne dollar américain en chaînes euro, comme suit :

```
private function usdToEuro(...args):String
{
    var usd:String = args[1];
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.828017;
    var euro:Number = Number(usd) * exchangeRate;
    trace(usd, Number(usd), euro);
    const euroSymbol:String = String.fromCharCode(8364); // €
    return euro.toFixed(2) + " " + euroSymbol;
}
```

`args[1]` représente le groupe entre parenthèses capturé mis en correspondance par l'expression régulière `usdPrice`. Il s'agit de la portion numérique de la chaîne dollar américain, c'est-à-dire la quantité en dollar, sans le signe `$`. La méthode applique une conversion de taux de change et renvoie la chaîne résultante (avec un symbole de fin de ligne € au lieu du symbole $ placé à gauche).

Chapitre 11 : Utilisation de XML

ActionScript 3.0 inclut un groupe de classes basé sur la spécification ECMAScript pour XML (E4X) (ECMA-357 version 2). Ces classes comprennent une fonctionnalité puissante et facile à utiliser permettant de travailler avec des données XML. A l'aide d'E4X, vous pouvez développer un code avec des données XML plus rapidement qu'avec les techniques de programmation précédentes. En outre, le code que vous produisez est plus facile à lire.

Ce chapitre décrit comment utiliser E4X pour traiter des données XML.

Principes de base de XML

Introduction à l'utilisation de XML

XML est un moyen standard de représenter des informations structurées afin de permettre aux ordinateurs de les utiliser facilement et aux utilisateurs de les écrire et de les comprendre. XML est l'abréviation de eXtensible Markup Language. La norme XML est disponible à l'adresse www.w3.org/XML/.

XML offre une façon pratique et standard de classer des données, afin de faciliter leur lecture, leur accès et leur manipulation. XML utilise une arborescence et une structure de balises identique à HTML. Voici un exemple simple de données XML :

```
<song>
    <title>What you know?</title>
    <artist>Steve and the flubberblubs</artist>
    <year>1989</year>
    <lastplayed>2006-10-17-08:31</lastplayed>
</song>
```

Les données XML peuvent également être plus complexes, avec des balises imbriquées dans d'autres balises ainsi que des attributs et d'autres composants structurels. Voici un exemple plus complexe de données XML :

```
<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>
```

Vous remarquerez que ce document XML contient d'autres structures XML complètes (les balises `song` avec leurs enfants, par exemple). Il démontre également d'autres structures XML telles que des attributs (`tracknumber` et `length` dans les balises `song`), et des balises qui contiennent d'autres balises au lieu de données (la balise `tracks`, par exemple).

Bien démarrer avec XML

Si vous avez peu ou pas d'expérience avec XML, voici une brève description des aspects les plus courants des données XML. Les données XML sont écrites sous forme de texte brut, avec une syntaxe spécifique permettant d'organiser les informations en un format structuré. Généralement, un seul jeu de données XML est appelé un *document XML*. Dans le format XML, les données sont organisées en *éléments* (qui peuvent être des éléments de données uniques ou des conteneurs pour d'autres éléments) à l'aide d'une structure hiérarchique. Chaque document XML possède un élément comme niveau supérieur ou élément principal. Cet élément racine peut contenir une seule information, même s'il est plus probable qu'il contienne d'autres éléments qui, à leur tour, contiennent d'autres éléments, et ainsi de suite. Par exemple, ce document XML contient les informations relatives à un album musical :

```
<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>
```

Chaque élément se distingue par un ensemble de *balises*—le nom de l'élément placé entre chevrons (signes inférieur à et supérieur à). La balise de début, indiquant le début de l'élément, porte le nom de l'élément :

```
<title>
```

La balise de fin, qui marque la fin de l'élément, a une barre oblique avant le nom de l'élément:

```
</title>
```

Si un élément est vide, il peut être écrit comme élément vide (parfois appelé élément de fin automatique). Dans XML, cet élément:

```
<lastplayed/>
```

est identique à cet élément :

```
<lastplayed></lastplayed>
```

Outre le contenu de l'élément placé entre les balises de début et de fin, un élément peut comporter d'autres valeurs, appelées *attributs*, définies dans la balise de début de l'élément. Par exemple, cet élément XML définit un seul attribut appelé `length`, avec la valeur `"4:19"` :

```
<song length="4:19"></song>
```

Chaque élément XML possède un contenu qui est soit une valeur, soit un ou plusieurs éléments XML, ou rien du tout (pour un élément vide).

En savoir plus sur XML

Pour en savoir plus sur l'utilisation de XML, un grand nombre de livres et de ressources sont disponibles, ainsi que les sites Web suivants :

- Didacticiel XML W3Schools : http://w3schools.com/xml/
- XML.com : http://www.xml.com/
- Didacticiels XMLpitstop, listes de discussions, etc. : http://xmlpitstop.com/

Classes ActionScript pour utiliser XML

ActionScript 3.0 inclut plusieurs classes permettant d'utiliser des informations structurées XML. Les deux classes principales sont les suivantes :

- XML : représente un seul élément XML, qui peut être un document XML avec plusieurs enfants ou un élément à une seule valeur dans un document.
- XMLList : représente un ensemble d'éléments XML. Un objet XMLList est utilisé lorsque plusieurs éléments XML sont des frères (au même niveau, et contenus par le même parent, dans la hiérarchie du document XML). Par exemple, une occurrence de XMLList serait le moyen le plus facile d'utiliser cet ensemble d'éléments XML (probablement contenus dans un document XML):

  ```
  <artist type="composer">Fred Wilson</artist>
  <artist type="conductor">James Schmidt</artist>
  <artist type="soloist">Susan Harriet Thurndon</artist>
  ```

Pour des utilisations plus avancées impliquant des espaces de noms XML, ActionScript inclut également les classes Namespace et QName. Pour plus d'informations, consultez la section « Utilisation des espaces de noms XML » à la page 247.

Outre les classes intégrées permettant d'utiliser XML, ActionScript 3.0 comprend également plusieurs opérateurs offrant des fonctionnalités spécifiques pour accéder et manipuler des données XML. Cette approche d'utilisation de XML au moyen de ces classes et de ces opérateurs est appelée ECMAScript pour XML (E4X), comme défini par la spécification ECMA-357 version 2.

Tâches XML courantes

Lorsque vous utilisez XML dans ActionScript, il est probable que vous effectuiez les tâches suivantes :

- Construction de documents XML (ajout d'éléments et de valeurs)

- Accès à des éléments, valeurs et attributs XML
- Filtrage (recherche dans) d'éléments XML
- Passage en boucle sur un ensemble d'éléments XML
- Conversion de données entre des classes XML et la classe String
- Utilisation des espaces de noms XML
- Chargement de fichiers XML externes

Concepts importants et terminologie

La liste de référence suivante contient les termes importants utilisés dans ce chapitre :

- Elément : élément unique dans un document XML, identifié comme le contenu se trouvant entre une balise de début et une balise de fin (balises comprises). Les éléments XML peuvent contenir des données de texte ou d'autres éléments, ou bien être vides.
- Elément vide : élément XML qui ne contient aucun élément enfant. Les éléments vides sont souvent écrits en tant que balises de fin automatique (`<element/>`, par exemple).
- Document : structure XML unique. Un document XML peut contenir n'importe quel nombre d'éléments (ou peut être constitué d'un seul élément vide) ; cependant, un document XML doit avoir un seul élément de niveau supérieur qui contient tous les autres éléments dans le document.
- Nœud : autre nom pour un élément XML.
- Attribut : valeur nommée associée à un élément qui est écrite dans la balise de début de l'élément au format `attributename="valeur"`, plutôt que d'être écrite comme élément enfant séparé imbriqué dans l'élément.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester des exemples de code. Tous les codes de ce chapitre comprennent l'appel de la fonction `trace()`. Pour tester les codes de ce chapitre :

1. Créez un document Flash vide.
2. Sélectionnez une image-clé dans le scénario.
3. Ouvrez le panneau Actions et copiez le code dans le panneau Script.
4. Exécutez le programme en sélectionnant Contrôle >Tester l'animation.

   Les résultats de la fonction `trace()` s'affichent dans le panneau Sortie.

Ceci ainsi que d'autres techniques de test des codes sont décrits en détail à la rubrique « Test des exemples de code contenus dans un chapitre » à la page 36.

Approche E4X concernant le traitement XML

La spécification ECMAScript pour XML définit un ensemble de classes et de fonctionnalités permettant d'utiliser des données XML. Cet ensemble de classes et de fonctionnalités est connu sous le nom de *E4X*. ActionScript 3.0 intègre les classes E4X suivantes : XML, XMLList, QName et Namespace.

Les méthodes, propriétés et opérateurs des classes E4X sont conçus avec les objectifs suivants :

- Simplicité : lorsque cela est possible, E4X facilite l'écriture et la compréhension du code à utiliser avec des données XML.

- Cohérence : les méthodes et le raisonnement E4X sont cohérents avec eux-mêmes et avec d'autres parties d'ActionScript.
- Connaissance : vous manipulez des données XML avec des opérateurs bien connus tels que l'opérateur point (.).

Remarque : *ActionScript 2.0 proposait la classe XML. Dans ActionScript 3.0, cette classe a été renommée XMLDocument afin d'être compatible avec la classe XML d'ActionScript 3.0 faisant partie d'E4X. Dans ActionScript 3.0, les classes héritées (XMLDocument, XMLNode, XMLParser et XMLTag) sont comprises dans le package flash.xml, pour la prise en charge du contenu hérité principalement. Les nouvelles classes E4X sont des classes de base. Vous ne devez pas importer de package pour les utiliser. Ce chapitre ne décrit pas les classes XML d'ActionScript 2.0 héritées de façon détaillée. Pour plus d'informations sur ces classes, consultez le package flash.xml package du Guide de référence du langage et des composants ActionScript 3.0.*

Voici un exemple de manipulation de données avec E4X :

```
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>
```

Votre application charge souvent des données XML depuis une source externe telle qu'un service Web ou un flux RSS. Néanmoins, par souci de clarté, les exemples fournis dans ce chapitre affectent des données XML comme littéraux.

Comme l'indique le code suivant, E4X inclut des opérateurs intuitifs tels que les opérateurs point (.) et identifiant d'attribut (`@`) pour accéder aux propriétés et aux attributs dans l'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
```

Utilisez la méthode `appendChild()` pour affecter un nouveau nœud enfant à l'XML, comme l'indique le code suivant :

```
var newItem:XML =
    <item id="3">
        <menuName>medium cola</menuName>
        <price>1.25</price>
    </item>

myXML.appendChild(newItem);
```

Utilisez les opérateurs `@` et `.` non seulement pour lire des données mais également pour les affecter, comme dans l'exemple suivant :

```
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";
```

Utilisez une boucle `for` pour parcourir en boucle les nœuds de l'XML, comme suit :

```
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));
```

Objets XML

Un objet XML peut représenter un élément XML, un attribut, un commentaire, une instruction de traitement ou un élément de texte.

Un objet XML est classé soit dans la catégorie des objets ayant un *contenu simple*, soit dans celle des objets ayant un *contenu complexe*. Un objet XML ayant des nœuds enfant est considéré comme ayant un contenu complexe. Le contenu est considéré comme simple s'il correspond à un attribut, un commentaire, une instruction de traitement ou un nœud de texte.

Par exemple, l'objet XML suivant contient un contenu complexe, y compris un commentaire et une instruction de traitement :

```
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>
```

Comme l'indique l'exemple suivant, vous pouvez maintenant utiliser les méthodes `comments()` et `processingInstructions()` pour créer des objets XML, un commentaire et une instruction de traitement :

```
var x2:XML = x1.comments()[0];
var x3:XML = x1.processingInstructions()[0];
```

Propriétés XML

La classe XML a cinq propriétés statiques :

- Les propriétés `ignoreComments` et `ignoreProcessingInstructions` déterminent si les commentaires ou les instructions de traitement sont ignorés lorsque l'objet XML est analysé.
- La propriété `ignoreWhitespace` détermine si les espaces blancs sont ignorés dans les balises d'un élément et les expressions intégrées séparées uniquement par des espaces blancs.

- Les propriétés `prettyIndent`et `prettyPrinting` servent à formater le texte renvoyé par les méthodes `toString()` et `toXMLString()` de la classe XML.

Pour plus d'informations sur ces propriétés, consultez le Guide de référence du langage et des composants ActionScript 3.0.

Méthodes XML

Les méthodes suivantes vous permettent d'utiliser la structure hiérarchique des objets XML :

- `appendChild()`
- `child()`
- `childIndex()`
- `children()`
- `descendants()`
- `elements()`
- `insertChildAfter()`
- `insertChildBefore()`
- `parent()`
- `prependChild()`

Les méthodes suivantes vous permettent d'utiliser des attributs d'objet XML :

- `attribute()`
- `attributes()`

Les méthodes suivantes vous permettent d'utiliser des propriétés d'objet XML :

- `hasOwnProperty()`
- `propertyIsEnumerable()`
- `replace()`
- `setChildren()`

Les méthodes suivantes vous permettent d'utiliser des espaces de noms et des noms qualifiés :

- `addNamespace()`
- `inScopeNamespaces()`
- `localName()`
- `name()`
- `namespace()`
- `namespaceDeclarations()`
- `removeNamespace()`
- `setLocalName()`
- `setName()`
- `setNamespace()`

Les méthodes suivantes vous permettent d'utiliser et de déterminer certains types de contenu XML :

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `nodeKind()`
- `processingInstructions()`
- `text()`

Les méthodes suivantes vous permettent d'effectuer des conversions en chaînes et de formater des objets XML :

- `defaultSettings()`
- `setSettings()`
- `settings()`
- `normalize()`
- `toString()`
- `toXMLString()`

Il existe quelques méthodes supplémentaires :

- `contains()`
- `copy()`
- `valueOf()`
- `length()`

Pour plus d'informations sur ces méthodes, consultez le Guide de référence du langage et des composants ActionScript 3.0.

Objets XMLList

Une occurrence de XMLList représente un ensemble arbitraire d'objets XML. Elle peut contenir des documents XML complets, des fragments XML ou les résultats d'une requête XML.

Les méthodes suivantes vous permettent d'utiliser la structure hiérarchique des objets XMLList :

- `child()`
- `children()`
- `descendants()`
- `elements()`
- `parent()`

Les méthodes suivantes vous permettent d'utiliser des attributs d'objet XMLList :

- `attribute()`
- `attributes()`

Les méthodes suivantes vous permettent d'utiliser des propriétés XMLList :

- `hasOwnProperty()`
- `propertyIsEnumerable()`

Les méthodes suivantes vous permettent d'utiliser et de déterminer certains types de contenu XML :

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `processingInstructions()`
- `text()`

Les méthodes suivantes vous permettent d'effectuer des conversions en chaînes et de formater les objets XMLList :

- `normalize()`
- `toString()`
- `toXMLString()`

Il existe quelques méthodes supplémentaires :

- `contains()`
- `copy()`
- `length()`
- `valueOf()`

Pour plus d'informations sur ces méthodes, consultez le Guide de référence du langage et des composants ActionScript 3.0.

Pour un objet XMLList qui contient exactement un élément XML, vous pouvez utiliser toutes les propriétés et les méthodes de la classe XML car un XMLList avec un élément XML est traité comme un objet XML. Par exemple, dans le code suivant, étant donné que `doc.div` est un objet XMLList contenant un élément, vous pouvez utiliser la méthode `appendChild()` de la classe XML :

```
var doc:XML =
        <body>
            <div>
                <p>Hello</p>
            </div>
        </body>;
doc.div.appendChild(<p>World</p>);
```

Pour consulter la liste des méthodes et des propriétés XML, consultez la section « Objets XML » à la page 237.

Initialisation de variables XML

Vous pouvez affecter un littéral XML à un objet XML, comme suit :

```
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>
```

Comme indiqué dans le code suivant, vous pouvez également utiliser le constructeur `new` pour créer une occurrence d'un objet XML à partir d'une chaîne contenant des données XML :

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
                            + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

Si les données XML dans la chaîne ne sont pas bien formées (par exemple, s'il manque une balise de fin), une erreur d'exécution se produit.

Vous pouvez également transmettre des données par référence (à partir d'autres variables) dans un objet XML, comme indiqué dans l'exemple suivant :

```
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>
```

Pour charger des données XML depuis une URL, utilisez la classe URLLoader, comme indiqué dans l'exemple suivant :

```
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!");
    }
}
```

Pour lire des données XML depuis une connexion de socket, utilisez la classe XMLSocket . Pour plus d'informations, consultez la classe XMLSocket du Guide de référence du langage et des composants ActionScript 3.0.

Assemblage et transformation d'objets XML

Utilisez la méthode `prependChild()` ou `appendChild()` pour ajouter une propriété au début ou à la fin de la liste des propriétés d'un objet XML, comme indiqué dans l'exemple suivant :

```
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>
```

Utilisez la méthode `insertChildBefore()` ou `insertChildAfter()` pour ajouter une propriété avant ou après une propriété spécifiée, comme suit :

```
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>)
```

Comme indiqué dans l'exemple suivant, vous pouvez également utiliser des accolades ({ et }) pour transmettre des données par référence (à partir d'autres variables) lorsque vous construisez des objets XML :

```
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)
}
```

Vous pouvez affecter des propriétés et des attributs à un objet XML à l'aide de l'opérateur `=`, comme dans l'exemple suivant :

```
var x:XML =
    <employee>
        <lastname>Smith</lastname>
    </employee>
x.firstname = "Jean";
x.@id = "239";
```

Ceci définit l'objet XML `x` de la façon suivante :

```
<employee id="239">
    <lastname>Smith</lastname>
    <firstname>Jean</firstname>
</employee>
```

Vous pouvez utiliser les opérateurs + et += pour concaténer des objets XMLList :

```
var x1:XML = <a>test1</a>
var x2:XML = <b>test2</b>
var xList:XMLList = x1 + x2;
xList += <c>test3</c>
```

Ceci définit l'objet XMLList `xList` de la façon suivante :

```
<a>test1</a>
<b>test2</b>
<c>test3</c>
```

Parcours de structures XML

L'une des fonctions puissantes d'XML est sa capacité à fournir des données imbriquées, complexes, via une chaîne linéaire de caractères de texte. Lorsque vous chargez des données dans un objet XML, ActionScript les analyse et charge sa structure hiérarchique en mémoire (ou envoie une erreur d'exécution si les données XML ne sont pas formées correctement).

Les opérateurs et les méthodes des objets XML et XMLList permettent de parcourir aisément la structure des données XML.

Utilisez l'opérateur point (.) et l'opérateur d'accesseur descendant (..) pour accéder aux propriétés enfant d'un objet XML. Considérez l'objet XML suivant :

```
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>
```

L'objet `myXML.book` est un objet XMLList contenant des propriétés enfant de l'objet `myXML` appelées `book`. Ces deux objets XML correspondent aux deux propriétés `book` de l'objet `myXML`.

L'objet `myXML..lastName` est un objet XMLList contenant des propriétés descendantes appelées `lastName`. Ces deux objets XML correspondent aux deux propriétés `lastName` de l'objet `myXML`.

L'objet `myXML.book.editor.lastName` est un objet XMLList contenant tout enfant appelé `lastName` des enfants appelés `editor` des enfants appelés `book` de l'objet `myXML` : en l'occurrence, un objet XMLList contenant un seul objet XML (la propriété `lastName` dont la valeur correspond à « `Case` »).

Accès aux nœuds enfant et parent

La méthode `parent()` renvoie le parent d'un objet XML.

Vous pouvez utiliser les valeurs d'index ordinales d'une liste enfant pour accéder à des objets enfant spécifiques. Par exemple, considérez un objet XML `myXML` ayant deux propriétés enfant appelées `book`. Chaque propriété enfant appelée `book` possède un numéro d'index qui lui est associé :

```
myXML.book[0]
myXML.book[1]
```

Pour accéder à un petit-enfant spécifique, vous pouvez indiquer des numéros d'index pour les noms de l'enfant et du petit-enfant:

```
myXML.book[0].title[0]
```

Cependant, s'il n'existe qu'un seul enfant de `x.book[0]` nommé `title`, vous pouvez omettre la référence d'index, comme suit :

```
myXML.book[0].title
```

De même, s'il n'existe qu'un seul enfant book de l'objet `x` et que cet objet enfant possède un seul objet title, vous pouvez omettre les deux références d'index, de la façon suivante :

```
myXML.book.title
```

Vous pouvez utiliser la méthode `child()` pour accéder à des enfants dont le nom est basé sur une variable ou une expression, comme indiqué dans l'exemple suivant :

```
var myXML:XML =
        <order>
            <book>
                <title>Dictionary</title>
            </book>
        </order>;

var childName:String = "book";

trace(myXML.child(childName).title) // output: Dictionary
```

Accès à des attributs

Utilisez le symbole `@` (l'opérateur identifiant d'attribut) pour accéder aux attributs dans un objet XML ou XMLList, comme indiqué dans le code suivant :

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@id); // 6401
```

Vous pouvez utiliser le symbole de caractère générique `*` avec le symbole `@` pour accéder à tous les attributs d'un objet XML ou XMLList, comme dans le code suivant :

```
var employee:XML =
    <employee id="6401" code="233">
        <lastName>Wu</lastName>
        <firstName>Erin</firstName>
    </employee>;
trace(employee.@*.toXMLString());
// 6401
// 233
```

Vous pouvez utiliser la méthode `attribute()` ou `attributes()` pour accéder à un attribut spécifique ou à tous les attributs d'un objet XML ou XMLList, comme dans le code suivant :

```
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
```

Vous pouvez également utiliser la syntaxe suivante pour accéder à des attributs, comme indiqué dans l'exemple suivant :

```
employee.attribute("id")
employee["@id"]
employee.@["id"]
```

Ils sont tous équivalents à `employee.@id`. Cependant, la syntaxe `employee.@id` est préférable.

Filtrage par attribut ou valeur d'élément

Vous pouvez utiliser les opérateurs parenthèses— `(` et `)` —pour filtrer des éléments avec un nom d'élément spécifique ou une valeur d'attribut. Considérez l'objet XML suivant :

```
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>
```

Les expressions suivantes sont toutes valides :

- `x.employee.(lastName == "McGee")`—Il s'agit du deuxième nœud `employee`.
- `x.employee.(lastName == "McGee").firstName`—Il s'agit de la propriété `firstName` du deuxième nœud `employee`.

- `x.employee.(lastName == "McGee").@id`—Il s'agit de la valeur de l'attribut `id` du deuxième nœud `employee`.
- `x.employee.(@id == 347)`—Le premier nœud `employee`.
- `x.employee.(@id == 347).lastName`—Il s'agit de la propriété `lastName` du premier nœud `employee`.
- `x.employee.(@id > 300)`—Il s'agit d'un XMLList avec deux propriétés `employee`.
- `x.employee.(position.toString().search("analyst") > -1)`—Il s'agit d'un XMLList avec deux propriétés `position`.

Si vous tentez de filtrer en fonction d'attributs ou d'éléments qui n'existent pas, Flash® Player et Adobe® AIR™ renvoient une exception. Par exemple, la dernière ligne du code suivant génère une erreur car il n'existe aucun attribut `id` dans le deuxième élément `p` :

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(@id == '123'));
```

De même, la dernière ligne du code suivant génère une erreur car il n'existe aucune propriété `b` du deuxième élément `p` :

```
var doc:XML =
            <body>
                <p id='123'>Hello, <b>Bob</b>.</p>
                <p>Hello.</p>
            </body>;
trace(doc.p.(b == 'Bob'));
```

Pour éviter ces erreurs, vous pouvez identifier les propriétés ayant les éléments ou les attributs correspondants, à l'aide des méthodes `attribute()` et `elements()`, comme dans le code suivant :

```
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'));
```

Vous pouvez également utiliser la méthode `hasOwnProperty()`, comme dans le code suivant :

```
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'));
```

Utilisation de for..in et for each.. dans les instructions

ActionScript 3.0 propose les instructions `for..in` et `for each..in` pour permettre les itérations dans les objets XMLList. Par exemple, considérez l'objet XML suivant, `myXML`, et l'objet XMLList, `myXML.item`. L'objet XMLList, `myXML.item`, est constitué de deux nœuds `item` de l'objet XML.

```
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>;
```

L'instruction `for....in` permet de procéder à une itération sur un ensemble de noms de propriété dans un objet XMLList :

```
var total:Number = 0;
for (var pname:String in myXML.item)
{
    total += myXML.item.@quantity[pname] * myXML.item.price[pname];
}
```

L'instruction `for each....in` permet de procéder à une itération dans les propriétés de l'objet XMLList :

```
var total2:Number = 0;
for each (var prop:XML in myXML.item)
{
    total2 += prop.@quantity * prop.price;
}
```

Utilisation des espaces de noms XML

Les espaces de noms dans un objet (ou document) XML identifient le type de données que contient l'objet. Par exemple, lorsque vous envoyez et fournissez des données XML à un service Web qui utilise le protocole de messagerie SOAP, vous déclarez l'espace de noms dans la balise de début de l'XML :

```
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>;
```

L'espace de noms a un préfixe, `soap`, et un URI qui le définit, `http://schemas.xmlsoap.org/soap/envelope/`.

ActionScript 3.0 inclut la classe Namespace pour utiliser des espaces de noms XML. Pour l'objet XML de l'exemple précédent, vous pouvez utiliser la classe Namespace comme suit :

```
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";
```

La classe XML comprend les méthodes suivantes qui permettent d'utiliser les espaces de noms : `addNamespace()`, `inScopeNamespaces()`, `localName()`, `name()`, `namespace()`, `namespaceDeclarations()`, `removeNamespace()`, `setLocalName()`, `setName()` et `setNamespace()`.

La directive `default xml namespace` vous permet d'affecter un espace de noms par défaut pour des objets XML. Par exemple, dans l'exemple suivant, `x1` et `x2` ont le même espace de noms par défaut :

```
var ns1:Namespace = new Namespace("http://www.example.com/namespaces/");
default xml namespace = ns1;
var x1:XML = <test1 />;
var x2:XML = <test2 />;
```

Conversion de type XML

Vous pouvez convertir des objets XML et XMLList en valeurs String. De même, vous pouvez convertir des chaînes en objets XML et XMLList. Sachez que toutes les valeurs d'attribut, les noms et les valeurs de texte XML sont des chaînes. Les sections suivantes décrivent toutes ces formes de conversion de type XML.

Conversion d'objets XML et XMLList en chaînes

Les classes XML et XMLList contiennent les méthodes `toString()` et `toXMLString()`. La méthode `toXMLString()` renvoie une chaîne qui comprend la totalité des balises, des attributs, des déclarations d'espace de noms et du contenu de l'objet XML. Pour les objets XML ayant un contenu complexe (éléments enfant), la méthode `toString()` procède exactement comme la méthode `toXMLString()`. Pour les objets XML ayant un contenu simple (ceux qui contiennent un seul élément de texte), la méthode `toString()` renvoie uniquement le contenu de texte de l'élément, comme indiqué dans l'exemple suivant :

```
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
```

Si vous utilisez la méthode `trace()` sans spécifier `toString()` ou `toXMLString()`, les données sont converties à l'aide de la méthode `toString()` par défaut, comme indiqué dans le code suivant :

```
var myXML:XML =
    <order>
        <item id='1' quantity='2'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
    <order>;

trace(myXML.item[0].menuName);
    // burger
```

Lorsque vous utilisez la méthode `trace()` pour déboguer un code, vous pouvez utiliser la méthode `toXMLString()` de façon à ce que la méthode `trace()` génère des données plus complètes.

Conversion de chaînes en objets XML

Vous pouvez utiliser le constructeur `new XML()` pour créer un objet XML à partir d'une chaîne, comme suit :

```
var x:XML = new XML("<a>test</a>");
```

Si vous tentez de convertir une chaîne en XML à partir d'une chaîne qui représente un XML non valide ou un XML qui n'est pas formé correctement, une erreur d'exécution est renvoyée, comme suit :

```
var x:XML = new XML("<a>test"); // throws an error
```

Conversion de valeurs d'attribut, de noms et de valeurs de texte à partir de chaînes

Toutes les valeurs d'attribut XML, les noms et les valeurs de texte sont des types de données String que vous pouvez convertir en d'autres types de données. Par exemple, le code suivant utilise la fonction `Number()` pour convertir des valeurs de texte en nombres :

```
var myXML:XML =
                        <order>
                            <item>
                                <price>3.95</price>
                            </item>
                            <item>
                                <price>1.00</price>
                            </item>
                        </order>;

var total:XML = <total>0</total>;
myXML.appendChild(total);

for each (var item:XML in myXML.item)
{
    myXML.total.children()[0] = Number(myXML.total.children()[0])
                                                + Number(item.price.children()[0]);
}
trace(myXML.total); // 4.35;
```

Si ce code n'utilisait pas la fonction `Number()`, il interpréterait l'opérateur + comme l'opérateur de concaténation de chaîne, et la méthode `trace()` de la dernière ligne génèrerait les éléments suivants :

```
01.003.95
```

Lecture de documents XML externes

Vous pouvez utiliser la classe URLLoader pour charger des données XML depuis une URL. Pour utiliser le code suivant dans vos applications, remplacez la valeur `XML_URL` dans l'exemple par une URL valide :

```
var myXML:XML = new XML();
var XML_URL:String = "http://www.example.com/Sample3.xml";
var myXMLURL:URLRequest = new URLRequest(XML_URL);
var myLoader:URLLoader = new URLLoader(myXMLURL);
myLoader.addEventListener("complete", xmlLoaded);

function xmlLoaded(event:Event):void
{
    myXML = XML(myLoader.data);
    trace("Data loaded.");
}
```

Vous pouvez également utiliser la classe XMLSocket pour définir une connexion de socket XML asynchrone avec un serveur. Pour plus d'informations, consultez le Guide de référence du langage et des composants ActionScript 3.0.

Exemple : chargement de données RSS depuis Internet

L'exemple d'application RSSViewer présente plusieurs fonctions d'utilisation d'XML dans ActionScript, notamment :

- Utilisation de méthodes XML pour parcourir des données XML sous la forme d'un flux RSS.
- Utilisation de méthodes XML pour assembler des données XML sous la forme HTML à utiliser dans un champ de texte.

Le format RSS est largement utilisé pour diffuser des nouvelles via XML. Un fichier de données RSS simple peut avoir l'aspect suivant :

```
<?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>
```

L'application SimpleRSS lit les données RSS depuis Internet, analyse les données à la recherche de titres, de liens et de descriptions et renvoie ces données. La classe SimpleRSSUI fournit l'IU et appelle la classe SimpleRSS qui effectue le traitement XML.

Pour obtenir les fichiers d'application de cet exemple, voir www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application RSSViewer se trouvent dans le dossier Samples/RSSViewer. L'application se compose des fichiers suivants :

Fichier	Description
RSSViewer.mxml ou RSSViewer.fla	Le fichier d'application principal dans Flash (FLA) ou Flex (MXML).
com/example/programmingas3/rssViewer/RSSParser.as	Une classe qui contient des méthodes utilisant E4X pour parcourir des données (XML) RSS et générer une représentation HTML correspondante.
RSSData/ak.rss	Un exemple de fichier RSS. Cette application est définie pour lire des données RSS à partir du Web, sur un flux RSS Flex hébergé par Adobe. Néanmoins, vous pouvez facilement modifier l'application pour lire des données RSS depuis ce document qui utilise un schéma légèrement différent de celui du flux RSS Flex.

Lecture et analyse de données XML

La classe RSSParser comprend une méthode `xmlLoaded()` qui convertit les données RSS d'entrée, stockées dans la variable `rssXML`, en une chaîne contenant une sortie formatée HTML, `rssOutput`.

Vers le début de la méthode, le code définit l'espace de noms XML par défaut si les données RSS source comprennent un espace de noms par défaut :

```
if (rssXML.namespace("") != undefined)
{
    default xml namespace = rssXML.namespace("");
}
```

Les lignes suivantes parcourent ensuite en boucle le contenu des données XML source, en examinant chaque propriété descendante appelée `item` :

```
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);
}
```

Les trois premières lignes définissent simplement des variables de chaîne pour représenter les propriétés de titre, de description et de lien de la propriété `item` des données XML. La ligne suivante appelle la méthode `buildItemHTML()` pour obtenir des données HTML sous la forme d'un objet XMLList, à l'aide des trois nouvelles variables de chaîne comme paramètres.

Assemblage de données XMLList

Les données HTML (un objet XMLList) se présentent comme suit :

```
<b>itemTitle</b>
<p>
    itemDescription
    <br />
    <a href="link">
        <font color="#008000">More...</font>
    </a>
</p>
```

Les premières lignes de la méthode effacent l'espace de noms xml par défaut :

```
default xml namespace = new Namespace();
```

La directive `default xml namespace` a un domaine de niveau bloc de fonction. Cela signifie que le domaine de cette déclaration est la méthode `buildItemHTML()`.

Les lignes qui suivent assemblent l'objet XMLList en fonction des arguments de chaîne transmis à la fonction :

```
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;
```

Cet objet XMLList représente des données de chaîne adaptées à un champ de texte HTML d'ActionScript.

La méthode `xmlLoaded()` utilise la valeur de renvoi de la méthode `buildItemHTML()` et la convertit en une chaîne :

```
XML.prettyPrinting = false;
rssOutput = outXML.toXMLString();
```

Extraction du titre du flux RSS et envoi d'un événement personnalisé

La méthode `xmlLoaded()` définit une variable de chaîne `rssTitle`, en fonction des informations contenues dans les données XML RSS source:

```
rssTitle = rssXML.channel.title.toString();
```

Pour finir, la méthode `xmlLoaded()` génère un événement qui notifie l'application que les données sont analysées et disponibles :

```
            dataWritten = new Event("dataWritten", true);
```

Chapitre 12 : Gestion des événements

Un système de gestion des événements permet au programmeur de répondre aux actions de l'utilisateur et aux événements système de manière pratique. Le modèle d'événements ActionScript 3.0 n'est pas uniquement pratique, il respecte les normes en vigueur et s'intègre parfaitement aux listes d'affichage d'Adobe® Flash® Player et d'Adobe® AIR™. Ce modèle repose sur la spécification d'événements Document Object Model (DOM) de niveau 3, une architecture de gestion d'événements normalisée. Il constitue donc pour les programmeurs ActionScript un outil puissant et parfaitement intuitif.

Ce chapitre s'organise en cinq sections. Les deux premières fournissent des informations générales sur la gestion d'événements dans ActionScript. Les trois dernières sections décrivent les principaux concepts qui sous-tendent le modèle d'événements : le flux d'événements, l'objet événement et les écouteurs d'événement. Dans ActionScript 3.0, le système de gestion des événements interagit étroitement avec la liste d'affichage ; ce chapitre suppose que vous connaissiez les principes de base de cette liste. Pour plus d'informations, consultez le chapitre « Programmation de l'affichage » à la page 277.

Principes de base de la gestion des événements

Introduction à la gestion des événements

Vous pouvez concevoir un événement comme tout type d'action qui se produit dans votre fichier SWF et qui présente un intérêt pour vous en tant que programmeur. Par exemple, la plupart des fichiers SWF prennent en charge une certaine forme d'interaction, qu'il s'agisse d'une action aussi simple qu'un clic avec la souris ou d'une opération plus complexe, telle que l'acceptation et le traitement des données saisies dans un formulaire. Toute interaction de ce type dans le fichier SWF est considérée comme un événement. Des événements peuvent également se produire sans aucun interaction directe de l'utilisateur, par exemple lorsque le chargement des données depuis un serveur se termine ou qu'une caméra reliée devient active.

Dans ActionScript 3.0, tout événement est représenté par un objet événement, qui correspond à une occurrence de la classe Event ou de l'une de ses sous-classes. Le rôle d'un objet événement est non seulement de stocker des informations relatives à un événement spécifique, mais aussi de contenir des méthodes qui favorisent la manipulation de cet objet. Par exemple, lorsque Flash Player ou AIR détecte un clic de la souris, il crée un objet événement (une occurrence de la classe MouseEvent) qui représente cet événement particulier.

Après la création d'un objet événement, Flash Player ou AIR le *distribue*, ce qui signifie que l'objet événement est transmis à l'objet représentant la cible de l'événement. L'objet qui doit recevoir l'objet événement ainsi distribué est appelé *cible d'événement*. Par exemple, lorsqu'une caméra reliée devient active, Flash Player distribue un objet événement directement à la cible de l'événement, dans ce cas l'objet représentant la caméra. Toutefois, si la cible d'événement se trouve dans la liste d'affichage, l'objet événement est transmis tout au long de la hiérarchie de la liste d'affichage jusqu'à ce qu'il atteigne la cible en question. Dans certains cas, l'objet événement se « propage ».ensuite vers le haut de la hiérarchie de la liste d'affichage, selon le même cheminement. Cette traversée de la hiérarchie de la liste d'affichage correspond au *flux d'événements*.

Vous pouvez « écouter ».les objets événement de votre code grâce aux écouteurs d'événement. Les *écouteurs d'événement* sont des fonctions ou des méthodes que vous écrivez pour répondre aux différents événements. Pour garantir que le programme réagisse aux événements, vous devez ajouter des écouteurs d'événement soit à la cible d'événement, soit à l'un des objets de la liste d'affichage qui font partie du flux d'événements de l'objet événement.

Chaque fois que vous écrivez un code d'écouteur d'événement, il suit cette structure de base (les éléments en gras sont des espaces réservés que vous rempliriez pour votre cas particulier) :

```
function eventResponse(eventObject:EventType):void
{
    // Actions performed in response to the event go here.
}

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

Ce code a un double rôle. Tout d'abord, il définit une fonction, qui est une manière de spécifier les actions à exécuter en réponse à l'événement. Ensuite, il appelle la méthode `addEventListener()` de l'objet source, « inscrivant ».ainsi la fonction auprès de l'événement spécifié de sorte que lorsque l'événement survient, les actions de la fonction ont lieu. Lorsque l'événement se produit, la cible de l'événement vérifie sa liste de toutes les fonctions et méthodes enregistrées en tant qu'écouteurs d'événement. Elle appelle ensuite chacune d'entre elles, transmettant l'objet événement comme paramètre.

Vous devez apporter quatre modifications à ce code pour créer votre propre écouteur d'événement. Premièrement, vous devez remplacer le nom de la fonction par celui que vous souhaitez utiliser (ceci doit être modifié à deux endroits, là où le code indique **eventResponse**). Deuxièmement, vous devez spécifier le nom de la classe de l'objet événement qui est envoyé par l'événement que vous souhaitez écouter (**EventType** dans le code), et vous devez indiquer la constante pour l'événement en question (**EVENT_NAME** dans la liste). Troisièmement, vous devez appeler la méthode `addEventListener()` sur l'objet qui enverra l'événement (**eventTarget** dans ce code). Vous pouvez également modifier le nom de la variable utilisée comme paramètre de la fonction (**eventObject** dans ce code).

Tâches courantes de gestion des événements

Vous trouverez ci-dessous des tâches courantes de gestion des événements. Chacune d'entre elles est décrite dans ce chapitre :

- Ecrire un code pour répondre à des événements
- Empêcher qu'un code réponde à des événements
- Utiliser des objets event
- Utiliser un flux d'événements :
  - Identification des informations de flux d'événements
  - Arrêt du flux d'événements
  - Arrêt du comportement par défaut
- Envoyer des événements à partir de vos classes
- Créer un type d'événement personnalisé

Concepts importants et terminologie

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Comportement par défaut : certains événements sont liés à un comportement appelé comportement par défaut. Par exemple, lorsqu'un utilisateur tape du texte dans un champ, un événement de saisie de texte est déclenché. Le comportement par défaut de cet événement consiste à afficher le caractère tapé dans le champ de texte—mais vous pouvez annuler ce comportement par défaut (si vous ne souhaitez pas afficher le caractère tapé, par exemple).
- Distribuer : indiquer à des écouteurs d'événement qu'un événement a eu lieu.
- Evénement : quelque chose qui s'est produit sur un objet et que ce dernier peut indiquer à d'autres objets.

- Flux d'événements : lorsque des événements concernent un objet sur la liste d'affichage (un objet affiché à l'écran), tous les objets qui contiennent l'objet sont informés de l'événement et avertissent à leur tour leurs écouteurs d'événement. Ce processus commence avec la scène et se poursuit à travers la liste d'affichage jusqu'à l'objet réel où s'est produit l'événement. Il recommence ensuite avec la scène. Ce processus est appelé flux d'événements.
- Objet événement : un objet contenant des informations sur l'occurrence d'un événement particulier qui est envoyé à tous les écouteurs lorsqu'un événement est distribué.
- Cible d'événement : l'objet qui envoie un événement. Par exemple, si l'utilisateur clique sur un bouton situé dans un Sprite se trouvant dans la scène, tous ces objets envoient des événements mais c'est au niveau de la cible d'événement que se produit l'événement (le bouton cliqué, dans ce cas).
- Ecouteur : un objet ou une fonction qui s'est enregistré avec un objet, pour indiquer qu'il doit être averti lorsqu'un événement spécifique se produit.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester des exemples de code. Tous les codes de ce chapitre comprennent un appel de la fonction `trace()` pour tester les résultats du code. Pour tester les codes de ce chapitre :

1. Créez un document vide à l'aide de l'outil de programmation Flash.
2. Sélectionnez une image-clé dans le scénario.
3. Ouvrez le panneau Actions et copiez le code dans le panneau Script.
4. Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

   Les résultats des fonctions `trace()` des codes s'affichent dans le panneau Sortie.

Certains codes sont plus complexes et sont écrits sous la forme d'une classe. Pour tester ces exemples :

1. Créez un document vide à l'aide de l'outil de programmation Flash et enregistrez-le sur votre ordinateur.
2. Créez un fichier ActionScript et enregistrez-le dans le même répertoire que le document créé à l'étape 1. Le nom du fichier doit correspondre au nom de la classe du code. Par exemple, si le code définit une classe EventTest, enregistrez le fichier ActionScript sous le nom EventTest.as.
3. Copiez le code dans le fichier ActionScript et enregistrez le fichier.
4. Dans le document, cliquez sur une partie vide de la scène ou de l'espace de travail pour activer l'Inspecteur des Propriétés du document.
5. Dans l'Inspecteur des Propriétés, dans le champ Classe du document, saisissez le nom de la classe ActionScript que vous avez copiée du texte.
6. Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

   Les résultats de l'exemple s'affichent dans le panneau Sortie.

Ces techniques de test d'exemples de code sont expliquées plus en détail à la section « Test des exemples de code contenus dans un chapitre » à la page 36.

Variation de la gestion d'événements dans ActionScript 3.0 par rapport aux versions antérieures

En ce qui concerne la gestion des événements, la différence la plus évidente entre ActionScript 3.0 et les versions antérieures est qu'ActionScript 3.0 comprend un seul système de gestion des événements alors que les anciennes versions d'ActionScript en comptent plusieurs. Cette section commence par une présentation générale du fonctionnement de la gestion des événements dans les versions précédentes, puis étudie les nouveautés qu'apporte ActionScript 3.0 dans ce domaine.

Gestion des événements dans les versions précédentes d'ActionScript

Antérieurement à ActionScript 3.0, le langage ActionScript fournissait plusieurs méthodes de gestion des événements :

- Les gestionnaires d'événement `on()`, qui peuvent se placer directement sur des occurrences Button et MovieClip
- Les gestionnaires d'événement `onClipEvent()`, qui peuvent se placer directement sur des occurrences MovieClip
- Des propriétés de fonction de rappel, telles que `XML.onload` et `Camera.onActivity`
- Des écouteurs d'événement, que vous pouvez enregistrer à l'aide de la méthode `addListener()`
- La classe UIEventDispatcher, qui implémentait partiellement le modèle d'événements DOM

Chacun de ces mécanismes présente des avantages et des inconvénients. Les gestionnaires `on()` et `onClipEvent()` sont simples d'utilisation, mais compliquent la maintenance des projets car il peut s'avérer difficile de localiser le code placé directement sur les boutons ou les clips. Les fonctions de rappel sont également faciles à implémenter, mais imposent une limite d'une seule fonction de rappel par événement. L'implémentation des écouteurs d'événement est plus complexe : ils nécessitent non seulement la création d'un objet et d'une fonction d'écouteur, mais aussi l'enregistrement de l'écouteur auprès de l'objet qui génère l'événement. Bien qu'elle accroisse le temps système nécessaire, cette solution vous permet de créer plusieurs objets écouteur et de tous les enregistrer pour le même événement.

Dans ActionScript 2.0, le développement des composants engendrait un modèle d'événements encore différent. Ce nouveau modèle, caractérisé par la classe UIEventDispatcher, reposait sur un sous-ensemble de la spécification d'événements DOM. Ainsi, pour les développeurs accoutumés à la gestion des événements de composant, le passage au nouveau modèle d'événements d'ActionScript 3.0 se fera sans trop de difficultés.

Malheureusement, si l'on constate des recoupements entre les divers modèles d'événements, il existe aussi des différences. Par exemple, dans ActionScript 2.0, certaines propriétés, telles que `TextField.onChanged`, peuvent s'utiliser soit comme fonction de rappel, soit comme écouteur d'événement. Toutefois, la syntaxe qui permet d'enregistrer les objets écouteurs varie selon que vous utilisez l'une des six classes qui prennent en charge les écouteurs ou la classe UIEventDispatcher. Pour les classes Key, Mouse, MovieClipLoader, Selection, Stage et TextField, vous utilisez la méthode `addListener()`, mais pour la gestion des événements de composant, vous utilisez une méthode appelée `addEventListener()`.

La multiplicité des modèles de gestion d'événements a fait naître une autre complexité : l'étendue de la fonction de gestionnaire d'événement variait largement en fonction du mécanisme utilisé. En d'autres termes, la signification du mot-clé `this` n'était pas cohérente sur l'ensemble des systèmes de gestion d'événements.

Gestion d'événements dans ActionScript 3.0

ActionScript 3.0 utilise pour la première fois un modèle de gestion d'événements qui vient remplacer les nombreux mécanismes qui existaient dans les précédentes versions du langage. Le nouveau modèle d'événements repose sur la spécification d'événements de niveau 3 DOM (Document Object Model). Bien que le format de fichier SWF ne suive pas spécifiquement la norme DOM, il existe suffisamment de similitudes entre la liste d'affichage et la structure du DOM pour permettre l'implémentation de ce modèle d'événements. Un objet de la liste d'affichage est semblable à un noeud de la structure hiérarchique du DOM *; dans ce chapitre, les termes* objet de liste d'affichage *et* noeud sont d'ailleurs utilisés de façon interchangeable.

L'implémentation du modèle d'événements DOM dans Flash Player et AIR comprend un concept appelé « comportements par défaut ». Un *comportement par défaut* est une action que Flash Player ou AIR effectue comme conséquence normale de certains événements.

Comportements par défaut

Les développeurs se chargent normalement d'écrire le code qui permet de répondre aux événements. Dans certains cas, cependant, un comportement est si couramment associé à un événement que Flash Player ou AIR l'exécute automatiquement, sauf si le développeur ajoute du code pour annuler son exécution. Comme Flash Player ou AIR se livre automatiquement à cette opération, on parle de comportements par défaut.

Par exemple, lorsqu'un utilisateur entre du texte dans un objet TextField, il est si courant de voir s'afficher la saisie dans l'objet TextField en question que ce comportement est prédéfini dans Flash Player ou AIR. Si vous ne souhaitez pas conserver ce comportement par défaut, vous pouvez l'annuler à l'aide du système de gestion des événements. Lorsqu'un utilisateur entre du texte dans un objet TextField, Flash Player ou AIR crée une occurrence de la classe TextEvent afin de représenter cette saisie. Pour éviter que Flash Player ou AIR n'affiche le texte dans l'objet TextField, vous devez accéder à cette occurrence de TextEvent spécifique et appeler sa méthode `preventDefault()`.

Certains comportements par défaut ne peuvent être évités. Par exemple, Flash Player et AIR génèrent un objet MouseEvent lorsque l'utilisateur double-clique sur un mot dans un objet TextField. Le comportement par défaut, qui ne peut être évité, consiste à mettre en évidence le mot situé sous le curseur.

De nombreux types d'objets événement ne sont associés à aucun comportement par défaut. Par exemple, l'objet événement Connect, que Flash Player distribue lorsqu'une connexion réseau est établie, n'est associé à aucun comportement par défaut. La documentation de l'API relative à la classe Event et ses sous-classes fait l'inventaire de chaque type d'événement, décrit le comportement par défaut qui lui est éventuellement associé et indique si ce dernier peut être évité.

Il est important de comprendre que les comportements par défaut sont uniquement associés à des objets événements distribués par Flash Player ou AIR ; il n'en existe aucun pour les objets événements distribués via ActionScript par programmation. Par exemple, vous pouvez utiliser les méthodes de la classe EventDispatcher pour distribuer un objet événement du type `textInput`, mais cet objet ne sera associé à aucun comportement par défaut. En d'autres termes, Flash Player et AIR n'affichent aucun caractère dans un objet TextField en réponse à un événement `textInput` que vous avez distribué par programmation.

Nouveautés des écouteurs d'événement dans ActionScript 3.0

Pour les développeurs qui connaissent bien la méthode ActionScript 2.0 `addListener()`, il peut être utile de souligner les différences entre le modèle d'écouteur d'événement d'ActionScript 2.0 et le modèle d'événements d'ActionScript 3.0. La liste ci-après décrit les principales différences entre ces deux modèles d'événements :

- Pour ajouter des écouteurs d'événement dans ActionScript 2.0, vous utilisez, selon le cas, `addListener()` ou `addEventListener()`. Dans ActionScript 3.0, il faut utiliser `addEventListener()` dans tous les cas.

- ActionScript 2.0 ne propose aucun flux d'événements dans ActionScript 2.0, ce qui signifie que la méthode `addListener()` peut uniquement être appelée sur l'objet qui émet l'événement. Dans ActionScript 3.0, la méthode `addEventListener()` peut être appelée sur tout objet faisant partie du flux d'événements.
- Dans ActionScript 2.0, les écouteurs d'événement peuvent être des fonctions, des méthodes ou des objets, alors que dans ActionScript 3.0, seules les fonctions et les méthodes peuvent agir comme écouteurs d'événement.

Flux d'événements

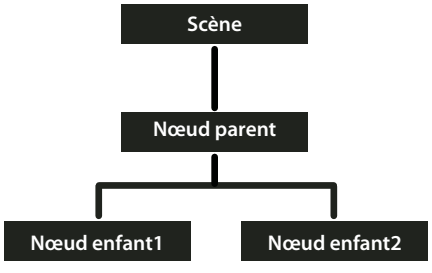
Flash Player ou AIR distribue des objets événements dès que survient un événement. Si la cible d'événement ne se trouve pas dans la liste d'affichage, Flash Player ou AIR distribue l'objet événement directement à la cible. Par exemple, Flash Player distribue l'objet événement Progress directement à un objet URLStream. Cependant, si la cible d'événement se trouve dans la liste d'affichage, Flash Player distribue l'objet événement à la liste d'affichage, dans laquelle l'objet chemine jusqu'à atteindre la cible d'événement.

Le *flux d'événements* représente le parcours que suivra un objet événement dans la liste d'affichage. Cette liste s'organise de manière hiérarchique, pour constituer une arborescence. Au sommet de la liste d'affichage se trouve la scène, un conteneur d'objet d'affichage spécial qui lui sert de racine. La Scène, représentée par la classe flash.display.Stage, est uniquement accessible via un objet d'affichage. Chaque objet d'affichage présente une propriété appelée `stage`, qui renvoie à la scène de cette application.

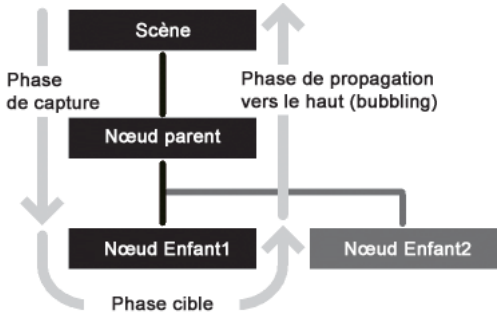
Lorsque Flash Player ou AIR distribue un objet d'événement pour un événement associé à une liste d'affichage, celui-ci effectue un aller-retour entre la Scène et le *noeud cible*. Selon la définition de la spécification d'événements DOM, le noeud cible est le nœud qui représente la cible d'événement. En d'autres termes, le noeud cible est l'objet de la liste d'affichage au niveau duquel est survenu l'événement. Par exemple, si l'utilisateur clique sur un objet de la liste d'affichage appelé `child1`, Flash Player ou AIR distribue un objet événement dont le noeud cible est `child1`.

Le flux d'événements se décompose en trois phases. La première correspond à la phase de capture, qui comprend tous les noeuds de la Scène jusqu'au parent du noeud cible. La deuxième partie est appelée la phase cible, qui comprend uniquement le noeud cible. La troisième partie s'appelle la phase de propagation vers le haut. Elle comprend les nœuds rencontrés lors du cheminement du parent du nœud cible jusqu'à la scène.

Le nom de ces phases prend tout son sens si vous envisagez la liste d'affichage comme une hiérarchie verticale dont le sommet est la Scène, comme illustré par le schéma suivant :



Si un utilisateur clique sur `Child1 Node`, Flash Player ou AIR distribue un objet événement dans ce flux d'événements. Comme le montre l'illustration suivante, le parcours de l'objet commence à `Stage`. L'objet descend ensuite jusqu'à `Parent Node`, puis vers `Child1 Node`. Il se propage alors vers le haut jusqu'à `Stage`, en repassant par `Parent Node`.



Dans cet exemple, la phase de capture comprend `Stage` et `Parent Node` pendant le trajet descendant initial. La phase cible comprend le temps passé au nœud `Child1 Node`. La phase de propagation comprend les nœuds `Parent Node` et `Stage`, qui se trouvent sur le chemin du retour vers le nœud racine.

Le flux d'événements contribue au renforcement du système de gestion des événements par rapport aux versions précédentes d'ActionScript. Dans ces dernières, le flux d'événements est inexistant, ce qui signifie que les écouteurs d'événement s'ajoutent uniquement à l'objet qui génère l'événement. Dans ActionScript 3.0, vous pouvez ajouter des écouteurs d'événement aussi bien à un noeud cible qu'à tout autre noeud du flux d'événements.

Cette possibilité d'ajouter des écouteurs d'événement tout au long du flux d'événements s'avère particulièrement utile lorsqu'un composant d'interface comprend plusieurs objets. Par exemple, un objet bouton contient souvent un objet texte qui sert de libellé au bouton. Sans la possibilité d'ajouter un écouteur au flux d'événements, il faudrait en ajouter un à l'objet bouton et un à l'objet texte pour être sûr d'être averti des événements de clic survenant à tout endroit du bouton. Le flux d'événements vous permet, au contraire, de placer un seul écouteur d'événement sur l'objet bouton afin de gérer les événements de clic, qu'ils se produisent sur l'objet texte ou sur des zones de l'objet bouton non couvertes par l'objet texte.

Cependant, certains objets événements ne participent pas aux trois phases du flux d'événements. Certains types d'événements, tels que `enterFrame` et `init`, sont distribués directement au nœud cible et ne participent ni à la phase de capture, ni à la phase de propagation vers le haut. D'autres événements peuvent cibler des objets qui ne font pas partie de la liste d'affichage, par exemple les événements distribués à une occurrence de la classe Socket. Ces objets événements aboutissent directement à l'objet cible, sans participer à la phase de capture et de propagation vers le haut.

Pour savoir comme se comporte un type d'événement particulier, vous pouvez consulter la documentation de l'API ou examiner les propriétés de l'objet événement. Cette dernière méthode est décrite à la section suivante.

Objets événement

Les objets événements jouent deux rôles essentiels dans le nouveau système de gestion des événements. Tout d'abord, ces objets représentent de véritables événements puisqu'ils stockent dans un ensemble de propriétés des informations relatives à des événements précis. Ils contient en outre un jeu de méthodes qui vous permet de manipuler les objets événements et d'agir sur le comportement du système de gestion des événements.

Pour faciliter l'accès à ces propriétés et ces méthodes, l'API Flash Player définit une classe Event qui constitue la classe de base de tous les objets événements. La classe Event définit un jeu fondamental de propriétés et de méthodes commun à tous les objets événements.

Cette section commence par étudier les propriétés de la classe Event avant de décrire les méthodes de cette même classe, puis explique l'existence de sous-classes dans la classe Event.

Présentation des propriétés de la classe Event

La classe Event définit plusieurs propriétés et constantes en lecture seule qui fournissent des informations essentielles sur l'objet événement. Les points suivants revêtent une importance particulière :

- Les types d'objet événement sont représentés par des constantes et stockés dans la propriété `Event.type`.
- La possibilité d'éviter le comportement par défaut d'un événement est représentée par une valeur booléenne, stockée dans la propriété `Event.cancelable`.
- Les informations relatives au flux d'événements se trouvent dans les propriétés restantes.

Types d'objets événement

Chaque objet événement est associé à un type d'événement. Les types d'événement sont stockés dans la propriété `Event.type` sous forme de chaîne. Il est utile de connaître le type d'un objet événement car votre code peut alors distinguer les objets de types différents. Par exemple, le code suivant spécifie que la fonction `clickHandler()` doit répondre à tous les objets événements clic de souris transmis à `myDisplayObject` :

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

La classe Event est elle-même associée à deux douzaines de types d'événement, représentés par des constantes de la classe Event. Dans cet extrait de la définition de la classe Event, certaines de ces constantes sont illustrées :

```
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
    }
}
```

Ces constantes permettent de faire facilement référence à des types d'événement précis. Vous devez utiliser ces constantes au lieu des chaînes qu'elles représentent. Si vous orthographiez de manière incorrecte un nom de constante dans votre code, le compilateur peut détecter l'erreur. Si vous utilisez les chaînes qu'elles représentent, une erreur de frappe ne sera pas forcément détectée lors de la compilation et pourrait provoquer un comportement inattendu, difficile à déboguer. Par exemple, utilisez le code suivant pour ajouter un écouteur d'événement :

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

plutôt que :

```
myDisplayObject.addEventListener("click", clickHandler);
```

Informations de comportement par défaut

Le code que vous écrivez est en mesure de vérifier si le comportement par défaut d'un objet événement donné peut être évité. Pour ce faire, il doit accéder à la propriété `cancelable`. La propriété `cancelable` contient une valeur booléenne qui indique si le comportement par défaut peut être évité ou non. Vous pouvez éviter, ou annuler, le comportement par défaut de quelques événements à l'aide de la méthode `preventDefault()`. Pour plus d'informations, consultez Annulation du comportement d'événement par défaut à la section « Présentation des méthodes de la classe Event » à la page 263.

Informations de flux d'événements

Les propriétés restantes de la classe Event contiennent des informations importantes sur l'objet événement et ses relations au flux d'événements, comme l'explique la liste suivante :

- La propriété `bubbles` contient des informations sur les parties du flux d'événements auquel participe l'objet événement.
- La propriété `eventPhase` indique la phase actuelle du flux d'événements.
- La propriété `target` stocke une référence à la cible d'événement.
- La propriété `currentTarget` stocke une référence de l'objet de liste d'affichage qui traite actuellement l'objet événement.

La propriété bubbles

On dit d'un événement qu'il se propage vers le haut (en anglais, « to bubble ») lorsqu'il participe à la phase de propagation vers le haut du flux d'événements, c'est-à-dire quand l'objet événement est transmis du nœud cible via ses ascendants jusqu'à la Scène. La propriété `Event.bubbles` stocke une valeur booléenne qui indique si l'objet événement participe à la phase de propagation vers le haut. Tous les événements qui se propagent vers le haut participent également aux phases de capture et cible ; de tels événements participent donc aux trois phases du flux d'événements. Si la valeur est `true`, l'objet événement participe aux trois phrases. Si la valeur est `false`, l'objet événement ne participe pas à la phase de propagation vers le haut.

La propriété eventPhase

Vous pouvez déterminer la phase d'événement de tout objet événement grâce à sa propriété `eventPhase`. La propriété `eventPhase` a pour valeur un entier non signé qui représente l'une des trois phases du flux d'événements. L'API de Flash Player définit une classe EventPhase distincte qui contient trois constantes correspondant aux trois valeurs entières non signées, comme illustré par l'extrait de code suivant :

```
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;
    }
}
```

Ces constantes correspondent aux trois valeurs valables pour la propriété `eventPhase`. Vous pouvez utiliser ces constantes pour améliorer la lisibilité de votre code. Supposons par exemple que vous souhaitiez être sûr qu'une fonction appelée `myFunc()` soit uniquement appelée lorsque la cible d'événement se trouve dans la scène cible. Le code suivant vous permet de tester cette condition :

```
if (event.eventPhase == EventPhase.AT_TARGET)
{
    myFunc();
}
```

La propriété target

La propriété `target` contient une référence à l'objet cible de l'événement. Dans certains cas, ce système est simple, par exemple, lorsqu'un micro devient actif, la cible de l'objet événement est l'objet Microphone. Toutefois, si la cible se trouve sur la liste d'affichage, il faut tenir compte de la hiérarchie de cette dernière. Par exemple, si un utilisateur clique avec la souris sur un point correspondant à plusieurs objets de la liste d'affichage qui se chevauchent, Flash Player et AIR choisissent toujours comme cible d'événement l'objet qui se trouve le plus loin de la Scène.

Dans des fichiers SWF complexes, et particulièrement ceux dont les boutons sont régulièrement ornés d'objets enfant plus petits, la propriété `target` ne doit pas être utilisée fréquemment car elle pointera souvent vers l'objet enfant du bouton plutôt que vers le bouton lui-même. Dans de telles situations, il est courant d'ajouter des écouteurs d'événement au bouton et d'utiliser la propriété `currentTarget`. En effet, cette dernière pointe vers le bouton alors que la propriété `target` peut pointer vers l'un des enfants du bouton.

La propriété currentTarget

La propriété `currentTarget` contient une référence de l'objet de liste d'affichage qui traite actuellement l'objet événement. Même s'il peut paraître étrange de ne pas savoir quel nœud traite actuellement l'objet événement que vous étudiez, gardez à l'esprit que vous pouvez ajouter une fonction écouteur à n'importe quel objet d'affichage du flux d'événements de l'objet événement en question. En outre, cette fonction écouteur peut être placée à tout endroit. Par ailleurs, la même fonction écouteur peut être ajoutée à différents objets d'affichage. L'utilité de la propriété `currentTarget` augmente donc avec la taille et la complexité du projet.

Présentation des méthodes de la classe Event

Il existe trois catégories de méthodes dans la classe Event :

- Les méthodes d'utilitaire, qui peuvent créer des copies d'un objet événement ou le convertir en chaîne
- Les méthodes de flux d'événements, qui suppriment les objets événements du flux d'événements
- Les méthodes de comportement par défaut, qui évitent le comportement par défaut ou vérifient s'il peut être évité

Méthodes d'utilitaire de la classe Event

La classe Event compte deux méthodes d'utilitaire. La méthode `clone()` permet de créer des copies d'un objet événement. La méthode `toString()` permet de représenter sous forme de chaînes les propriétés d'un objet événement ainsi que leurs valeurs. Bien qu'utilisées en interne par le modèle d'événements, ces deux méthodes sont mises à la disposition des développeurs pour un usage générique.

Pour les développeurs expérimentés qui souhaitent créer des sous-classes de la classe Event, il est nécessaire de redéfinir et d'implémenter des versions de ces deux méthodes d'utilitaires afin de garantir le bon fonctionnement de la sous-classe d'événement.

Arrêt du flux d'événements

La méthode `Event.stopPropagation()` ou `Event.stopImmediatePropagation()` vous permet d'arrêter le cheminement d'un objet événement dans le flux d'événements. Quasi identiques, ces deux méthodes diffèrent uniquement en ce que les autres écouteurs d'événement du nœud actuel sont autorisés ou non à s'exécuter :

- La méthode `Event.stopPropagation()` empêche l'objet événement de passer au nœud suivant mais seulement après que tous les autres écouteurs du nœud actuel ont été autorisés à s'exécuter.
- La méthode `Event.stopImmediatePropagation()` empêche l'objet événement de passer au nœud suivant sans autoriser les autres écouteurs du nœud actuel à s'exécuter.

Quelle que soit la méthode appelée, elle n'a aucun effet sur la réalisation du comportement par défaut de l'événement. Utilisez les méthodes de comportement par défaut de la classe Event pour éviter le comportement par défaut.

Annulation du comportement d'événement par défaut

Deux méthodes sont liées à l'annulation du comportement par défaut : preventDefault() `et` isDefaultPrevented(). Appelez la méthode `preventDefault()` pour annuler le comportement par défaut associé à un événement. Pour vérifier si `preventDefault()` a déjà été appelée sur un objet événement, appelez la méthode `isDefaultPrevented()`, qui renvoie la valeur `true` si la méthode a déjà été appelée, `false` dans le cas contraire.

La méthode `preventDefault()` fonctionne uniquement s'il est possible d'annuler le comportement par défaut de l'événement. Pour vérifier que c'est le cas, reportez-vous à la documentation de l'API de ce type d'événement ou examinez la propriété `cancelable` de l'objet événement à l'aide du code ActionScript.

L'annulation du comportement par défaut n'a aucun effet sur la progression d'un objet événement dans le flux d'événements. Utilisez les méthodes de flux d'événements de la classe Event pour supprimer un objet événement du flux d'événements.

Sous-classes de la classe Event

Pour de nombreux événements, le jeu de propriétés commun, défini dans la classe Event est suffisant. Néanmoins, d'autres événements présentent des caractéristiques exclusives qui ne peuvent être capturées par les propriétés disponibles dans la classe Event. Pour ces événements, ActionScript 3.0 définit plusieurs sous-classes de la classe Evénement.

Chaque sous-classe fournit un complément de propriétés et de types d'événement spécifiques à la catégorie d'événement considérée. Par exemple, les événements liés aux actions de la souris présentent plusieurs caractéristiques uniques, que les propriétés définies dans la classe Event ne peuvent capturer. La classe MouseEvent constitue une extension de la classe Event puisqu'elle ajoute dix propriétés contenant des informations telles que l'emplacement de l'événement de souris et les éventuelles touches actionnées en même temps.

Une sous-classe d'Event contient également des constantes qui représentent de types d'événement associés à la sous-classe. Par exemple, la classe MouseEvent définit des constantes pour plusieurs types d'événement de souris, notamment `click`, `doubleClick`, `mouseDown` et `mouseUp`.

Comme le décrit la section consacrée aux méthodes d'utilitaire de la classe Event dans « Objets événement » à la page 260, lors de la création d'une sous-classe d'Event, vous devez bloquer les méthodes `clone()` et `toString()` pour fournir la fonctionnalité propre à la sous-classe.

Les écouteurs d'événement

Les écouteurs d'événement, également appelés gestionnaires d'événements, sont des fonctions que Flash Player et AIR exécutent en réponse à des événements déterminés. La procédure d'ajout d'un écouteur d'événement se déroule en deux temps. En premier lieu, vous créez une fonction ou méthode de classe que Flash Player ou AIR doit exécuter en réponse à l'événement. On parle parfois de fonction d'écouteur ou de fonction de gestionnaire d'événement. En second lieu, vous utilisez la méthode `addEventListener()` pour enregistrer la fonction d'écouteur auprès de la cible de l'événement ou tout autre objet de la liste d'affichage qui appartient au flux d'événements approprié.

Création d'une fonction d'écouteur

La création d'une fonction d'écouteur est un domaine dans lequel le modèle d'événements ActionScript 3.0 diffère du modèle d'événements DOM. Dans le modèle d'événements DOM, on distingue clairement un écouteur d'événement et une fonction d'écouteur : un écouteur d'événement est une occurrence de classe qui implémente l'interface EventListener, tandis qu'une fonction d'écouteur est une méthode de cette classe appelée `handleEvent()`. Dans le modèle d'événements DOM, vous enregistrez l'occurrence de classe qui contient la fonction d'écouteur, plutôt que la fonction d'écouteur elle-même.

Le modèle d'événements ActionScript ne fait aucune distinction entre l'écouteur d'événement et la fonction d'écouteur. L'interface EventListener est inexistante dans ActionScript 3.0 et les fonctions d'écouteur peuvent être définies en dehors de toute classe ou au sein d'une classe. Par ailleurs, il n'est pas nécessaire de nommer les fonctions d'écouteur `handleEvent()` ; vous pouvez utiliser tout identifiant valable. Dans ActionScript 3.0, vous enregistrez le nom de la fonction d'écouteur elle-même.

Fonction d'écouteur définie en dehors de toute classe

Le code suivant crée un fichier SWF simple qui affiche une forme carrée de couleur rouge. Une fonction d'écouteur appelée `clickHandler()`, qui n'appartient à aucune classe, écoute les événements de clic de souris dans le carré rouge.

```
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);
}
```

Lorsqu'un utilisateur interagit avec le fichier SWF résultant, en cliquant sur le carré, Flash Player ou AIR génère la sortie de trace ci-après :

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

Notez que l'objet événement est transmis sous forme d'instruction à `clickHandler()`. Cela permet à votre fonction d'écouteur d'examiner l'objet événement. Dans cet exemple, vous utilisez la propriété `type` de l'objet événement pour vérifier que cet événement correspond à un clic.

L'exemple vérifie aussi la valeur du mot-clé `this`. Dans ce cas, `this` représente l'objet global, ce qui est logique puisque la fonction est définie en dehors de toute classe ou objet personnalisé.

Fonction d'écouteur définie comme méthode de classe

L'exemple ci-dessous est identique au précédent, qui définit la classe ClickExample, sauf que la fonction `clickHandler()` est définie comme méthode de la classe 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);
    }
}
```

Lorsqu'un utilisateur interagit avec le fichier SWF résultant, en cliquant sur le carré rouge, Flash Player ou AIR génère la sortie de trace ci-après :

```
clickHandler detected an event of type: click
the this keyword refers to: [object ChildSprite]
```

Notez que le mot-clé `this` renvoie à l'occurrence de ChildSprite appelée `child`. Voici un changement de comportement par rapport à ActionScript 2.0. Si vous utilisiez des composants dans ActionScript 2.0, vous vous rappelez sans doute que lorsqu'une méthode de classe était transmise à `UIEventDispatcher.addEventListener()`, l'étendue de la méthode était liée au composant qui émettait l'événement, et non à la classe dans laquelle la méthode d'écouteur était définie. En d'autres termes, si vous utilisiez cette technique dans ActionScript 2.0, le mot-clé `this` renvoyait au composant émettant l'événement et non à l'occurrence de ChildSprite.

Pour certains développeurs, il s'agissait d'un vrai problème car cela signifiait qu'ils ne pouvaient accéder à aucune autre méthode et propriété de la classe qui contenait la méthode d'écouteur. Pour le contourner, les programmeurs d'ActionScript 2.0 pouvaient utiliser la classe `mx.util.Delegate` pour modifier l'étendue de la méthode d'écouteur. Cette manipulation n'est plus nécessaire puisque ActionScript 3.0 crée une méthode liée lorsque `addEventListener()` est appelée. Par conséquent, le mot-clé `this` fait référence à l'occurrence de ChildSprite appelée `child` et le programmeur peut accéder aux autres méthodes et propriétés de la classe ChildSprite.

Ecouteur d'événement à ne pas utiliser

Une troisième technique permet de créer un objet générique dont l'une des propriétés pointe vers une fonction d'écouteur affectée dynamiquement. Elle est cependant déconseillée. Nous l'évoquons ici en raison de son utilisation courante dans ActionScript 2.0 ; il n'est toutefois pas recommandé de l'utiliser dans ActionScript 3.0. Cette mise en garde tient au fait que le mot-clé `this` fera référence à l'objet global et non à l'objet écouteur.

L'exemple ci-après est identique à l'exemple précédent de la classe ClickExample, sauf que la fonction d'écouteur est définie comme faisant partie d'un objet générique appelé `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);
}
```

Les résultats de trace seront les suivants :

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

On s'attendrait à ce que `this` fasse référence à `myListenerObj` et que la sortie de trace soit `[object Object]`, mais le mot-clé renvoie en fait à l'objet global. Lorsque vous transmettez un nom de propriété dynamique comme instruction à `addEventListener()`, Flash Player ou AIR est incapable de créer une méthode liée. En effet, ce que vous transmettez comme paramètre `listener` n'est rien de plus que l'adresse mémoire de votre fonction d'écouteur ; `Flash Player et AIR n'ont aucun moyen de lier cette adresse à l'occurrence de` myListenerObj.

Gestion des écouteurs d'événement

Vous pouvez gérer vos fonctions d'écouteur à l'aide des méthodes de l'interface IEventDispatcher. Cette interface est la version ActionScript 3.0 de l'interface EventTarget du modèle d'événements DOM. Bien que le nom IEventDispatcher semble impliquer que l'objet principal de la classe est l'envoi (ou la distribution) des objets événements, les méthodes qui lui correspondent servent en fait plus souvent à l'enregistrement, la vérification et la suppression des écouteurs d'événement. L'interface IEventDispatcher définit cinq méthodes, comme illustré dans le code suivant :

```
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;
    }
}
```

L'API de Flash Player implémente l'interface IEventDispatcher à l'aide de la classe Event Dispatcher. Cette dernière constitue la classe de base de toutes les classes pouvant servir de cibles d'événement ou faire partie d'un flux d'événements. Par exemple, la classe DisplayObject hérite de la classe EventDispatcher, par conséquent, tout objet de la liste d'affichage peut accéder aux méthode de l'interface IEventDispatcher.

Ajout des écouteurs d'événement

La méthode `addEventListener()` est la clé de voûte de l'interface IEventDispatcher. Elle permet d'enregistrer les fonctions d'écouteurs. Les deux paramètres requis sont `type` et `listener`. Le paramètre `type` spécifie le type d'événement. Avec le paramètre `listener`, vous pouvez spécifier la fonction d'écouteur qui doit s'exécuter lorsque l'événement survient. Le paramètre `listener` peut être une référence à une fonction ou une méthode de classe.

Remarque : *n'utilisez pas de parenthèses lors de la spécification du paramètre `listener`. Par exemple, la fonction `clickHandler()` est spécifiée sans parenthèses dans l'appel suivant à la méthode `addEventListener()` :*

Remarque : *`addEventListener(MouseEvent.CLICK, clickHandler).`*

Le paramètre `useCapture` de la méthode `addEventListener()` vous permet de contrôler la phase du flux d'événements pendant laquelle votre écouteur sera actif. Si `useCapture` a la valeur `true`, votre écouteur sera actif pendant la phase de capture du flux d'événements. Si `useCapture` a la valeur `false`, votre écouteur sera actif pendant la phase cible et la phase de propagation du flux d'événements. Pour écouter un événement pendant toutes les phases du flux d'événements, vous devez appeler deux fois `addEventListener() ; la première fois` useCapture `prend la valeur` true, `la seconde,` useCapture `prend la valeur` false.

Le paramètre `priority` de la méthode `addEventListener()` ne fait pas officiellement partie du modèle d'événements DOM de niveau 3. Il est inclus dans ActionScript 3.0 pour vous offrir une plus grande souplesse dans l'organisation de vos écouteurs d'événement. Lorsque vous appelez `addEventListener()`, vous pouvez définir la priorité de cet écouteur d'événement en transmettant une valeur entière comme paramètre `priority`. La valeur par défaut est 0. Vous pouvez toutefois utiliser une valeur entière négative ou positive. Plus le nombre est élevé, plus l'exécution de l'écouteur d'événement est rapide. Les écouteurs d'événement de priorité équivalente sont exécutés suivant l'ordre dans lequel ils ont été ajoutés : plus l'écouteur est ajouté tôt, plus il est exécuté rapidement.

Le paramètre `useWeakReference` vous permet de spécifier si la référence à la fonction d'écouteur est faible ou normale. En lui attribuant la valeur `true`, vous évitez les situations dans lesquelles les fonctions d'écouteurs demeurent dans la mémoire alors qu'elles sont inutiles. Flash Player et AIR utilisent une technique appelée *nettoyage* pour effacer de la mémoire les objets qui ne servent plus. Un objet est considéré comme inutilisé lorsqu'il n'apparaît dans aucune référence. Le nettoyeur de mémoire ignore les références faibles, c'est-à-dire qu'une fonction d'écouteur vers laquelle pointe uniquement une référence faible est incluse dans le nettoyage.

Suppression des écouteurs d'événement

La méthode `removeEventListener()` permet de supprimer un écouteur d'événement dont vous n'avez plus besoin. Il est judicieux de supprimer tous les écouteurs qui ne seront plus utilisés. Les paramètres requis sont notamment `eventName` et `listener`, soit les mêmes que ceux requis pour la méthode `addEventListener()`. Rappel : pour écouter les événements pendant toutes les phases du flux d'événements, vous pouvez appeler `addEventListener()` deux fois, en attribuant à `useCapture` la valeur `true` la première, puis `false` la seconde. Pour supprimer les deux écouteurs d'événement, il serait nécessaire d'appeler `removeEventListener()` à deux reprises, la première fois en attribuant la valeur `true` à `useCapture`, la seconde fois en utilisant la valeur `false`.

Distribution d'événements

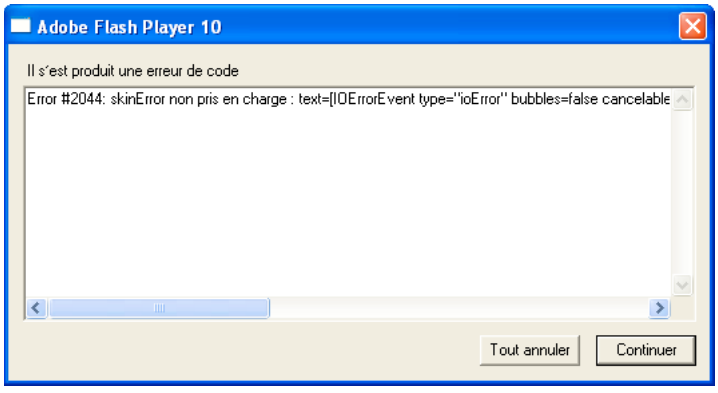
La méthode `dispatchEvent()` peut servir aux développeurs chevronnés pour distribuer un objet événement personnalisé dans le flux d'événements. Cette méthode accepte un seul paramètre, une référence à l'objet événement, qui doit être une occurrence de la classe Event ou de l'une de ces sous-classes. Après distribution, la propriété `target` de l'objet événement est définie avec l'objet sur lequel portait l'appel `dispatchEvent()`.

Vérification des écouteurs d'événement existants

Les deux dernières méthodes de l'interface IEventDispatcher fournissent des informations précieuses sur l'existence des écouteurs d'événement. La méthode `hasEventListener()` renvoie la valeur `true` si un écouteur d'événement est détecté pour un type d'événement spécifique sur un objet particulier de la liste d'affichage. La méthode `willTrigger()` renvoie également la valeur `true` si un écouteur est détecté pour un objet donné de la liste d'affichage. Cependant `willTrigger()` vérifie les écouteurs sur l'objet d'affichage en question mais également sur tous les ascendants de cet objet dans l'ensemble des phases du flux d'événements.

Evénements d'erreur sans écouteurs

Plus que les événements, les exceptions constituent le mécanisme principal de gestion des erreurs dans ActionScript 3.0. Toutefois, la gestion des exceptions ne fonctionne pas sur les opérations asynchrones telles que les chargements de fichiers. Si une erreur survient pendant une opération asynchrone, Flash Player et AIR distribuent un objet événement d'erreur. Si vous ne créez pas d'écouteur pour l'événement d'erreur, les versions de débogage de Flash Player et AIR affichent une boîte de dialogue comportant des informations sur l'erreur en question. Par exemple, la version de débogage de Flash Player affiche la boîte de dialogue suivante, qui décrit l'erreur associée à une tentative de chargement d'un fichier par l'application à partir d'une URL non valide :



La plupart des événements d'erreur reposent sur la classe ErrorEvent. Ils présentent donc une propriété appelée `text`, qui sert au stockage du message d'erreur que Flash Player ou AIR affiche. Il existe deux exceptions : les classes StatusEvent et NetStatusEvent. Ces deux classes possèdent une propriété `level` (`StatusEvent.level` et `NetStatusEvent.info.level`). Lorsque la valeur de la propriété `level` est `error`, ces types d'événement sont considérés comme des événements d'erreur.

Un événement d'erreur n'interrompt pas l'exécution du fichier SWF. Il se traduit uniquement par l'affichage d'une boîte de dialogue dans les versions de débogage des navigateurs et des lecteurs autonomes, d'un message dans le panneau de sortie du lecteur de création et d'une entrée dans le fichier journal d'Adobe Flex Builder 3. Aucune manifestation n'est visible dans les autres versions de Flash Player ou AIR.

Exemple : Alarm Clock

L'exemple Alarm Clock correspond à une horloge qui permet à l'utilisateur de déterminer l'heure à laquelle l'alarme doit se déclencher et d'afficher un message en même temps. Il repose sur l'application SimpleClock du chapitre « Utilisation des dates et des heures » à la page 135 et illustre de nombreux aspects de l'utilisation des événements dans ActionScript 3.0, notamment les suivants :

- Ecoute des événements et réponse
- Notification d'un événement aux écouteurs
- Créer un type d'événement personnalisé

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application Alarm Clock se trouvent dans le dossier Samples/AlarmClock. Il s'agit des fichiers suivants :

Fichier	Description
AlarmClockApp.mxml ou AlarmClockApp.fla	Le fichier d'application principal dans Flash (FLA) ou Flex (MXML).
com/example/programmingas3/clock/AlarmClock.as	Classe permettant d'étendre la classe SimpleClock, qui ajoute la fonctionnalité de réveil.
com/example/programmingas3/clock/AlarmEvent.as	Une classe d'événement personnalisé (sous-classe de flash.events.Event), qui sert d'objet événement à l'événement `alarm` de la classe AlarmClock.
com/example/programmingas3/clock/AnalogClockFace.as	Dessine une horloge ronde et les aiguilles des heures, des minutes et des secondes en fonction de l'heure (décrit dans l'exemple SimpleClock).
com/example/programmingas3/clock/SimpleClock.as	Composant d'interface d'horloge doté d'une fonctionnalité simple de mesure temporelle (décrit dans l'exemple SimpleClock).

Présentation du réveil

Dans cet exemple, la principale fonctionnalité de l'horloge (dont la mesure du temps et l'affichage du cadran) réutilise le code de l'application SimpleClock, décrite à la section « Exemple : horloge analogique simple » à la page 140. La classe AlarmClock étend la classe SimpleClock de cet exemple en y ajoutant la fonctionnalité de réveil requise : réglage de l'heure de déclenchement et avertissement une fois l'alarme déclenchée.

Le rôle des événements est de fournir un avertissement lorsque se produit quelque chose. La classe AlarmClock expose l'événement Alarme, à l'écoute duquel d'autres objets peuvent être placés afin d'effectuer les actions voulues. En outre, la classe AlarmClock utilise une occurrence de la classe Timer pour déterminer à quel moment déclencher l'alarme. Comme la classe AlarmClock, la classe Timer fournit un événement pour avertir d'autres objets (une occurrence de AlarmClock dans ce cas) une fois un certain délai écoulé. Comme dans la plupart des applications ActionScript, les événements constituent une part importante de la fonctionnalité de l'exemple Alarm Clock.

Déclenchement de l'alarme

Comme mentionné plus haut, la seule fonctionnalité de la classe AlarmClock est liée à la définition et au déclenchement de l'alarme. La classe intégrée Timer (flash.utils.Timer) permet au développeur de définir du code qui sera exécuté après un délai spécifique. La classe AlarmClock utilise une occurrence de Timer pour déterminer le moment auquel déclencher l'alarme.

```
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);
}
```

L'occurrence de Timer définie dans la classe AlarmClock est appelée `alarmTimer`. La méthode `initClock()`, qui effectue les opérations de configuration nécessaires à l'occurrence de AlarmClock, exploite la variable `alarmTimer` de deux manières. Tout d'abord, la variable est instanciée avec les paramètres indiquant à l'occurrence de Timer d'attendre 0 millisecondes et de déclencher l'événement timer une seule fois. Après instanciation de `alarmTimer`, le code appelle la méthode `addEventListener()` de cette variable pour indiquer qu'il veut écouter l'événement `timer` de cette variable. Le fonctionnement d'une occurrence de Timer repose sur la distribution de l'événement `timer` après un certain délai. La classe AlarmClock doit savoir quand l'événement `timer` est distribué afin de déclencher sa propre alarme. En appelant `addEventListener()`, le code AlarmClock s'enregistre comme écouteur auprès de `alarmTimer`. Les deux paramètres indiquent que la classe AlarmClock souhaite écouter l'événement `timer` (indiqué par la constante `TimerEvent.TIMER`), et que lorsque l'événement survient, la méthode `onAlarm()` de la classe AlarmClock doit être appelée en réponse à l'événement.

Pour effectivement définir l'alarme, la méthode `setAlarm()` de la classe AlarmClock est appelée, comme suit :

```
/**
 * 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;
}
```

Cette méthode effectue plusieurs opérations, notamment le stockage du message d'alarme et la création d'un objet Date (`alarmTime`) représentant le moment réel où l'alarme se déclenchera. Point le plus important de cette étude, le minuteur de la variable `alarmTimer`, dans les dernières lignes la méthode, est défini et activé. Tout d'abord, la méthode `reset()` est appelée, qui arrête le minuteur et le remet à zéro s'il est déjà reparti. Ensuite, l'heure actuelle (représentée par la variable `now`) est soustraite à la valeur de la variable `alarmTime` afin de déterminer combien de millisecondes doivent s'écouler avant le déclenchement de l'alarme. La classe Timer ne déclenche pas l'événement `timer` à une heure absolue `; c'est ce décalage relatif qui est attribué à la propriété` delay `de` alarmTimer. Enfin, la méthode `start()` est appelée pour lancer le minuteur.

Une fois le délai spécifié écoulé, `alarmTimer` distribue l'événement `timer.` Comme la classe AlarmClock s'est enregistrée comme écouteur auprès de sa méthode `onAlarm()` pour l'événement `timer`, lorsque celui-ci survient, `onAlarm()` est appelée.

```
/**
 * 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);
}
```

Lorsqu'une méthode est enregistrée comme écouteur d'événement, elle doit être définie avec la signature adaptée (c'est-à-dire le jeu de paramètres et le type de renvoi de la méthode). Pour écouter l'événement `timer` de la classe Timer, une méthode doit comporter un paramètre dont le type de données est TimerEvent (flash.event.TimerEvent), une sous-classe de la classe Event. Lorsque l'occurrence de Timer appelle ses écouteurs d'événement, elle transmet une occurrence de TimerEvent à l'objet événement.

Notification de l'alarme à d'autres composants

De même que la classe Timer, la classe AlarmClock fournit un événement qui permet de transmettre des notifications à d'autres éléments de code lorsque l'alarme se déclenche. Pour qu'une classe puisse utiliser le système de gestion des événements intégré à ActionScript, elle doit implémenter l'interface flash.events.IEventDispatcher. La plupart du temps, cela se fait par extension de la classe flash.events.EventDispatcher, qui assure une implémentation standard de IEventDispatcher (ou par extension de l'une des sous-classes de EventDispatcher). Comme décrit précédemment, la classe AlarmClock étend la classe SimpleClock, qui à son tour étend la classe Sprite, elle-même extension de la classe EventDispatcher (par héritage). Ainsi, la classe AlarmClock intègre déjà une fonctionnalité lui permettant de fournir ses propres événements.

D'autres éléments de code peuvent s'enregistrer pour être notifiés de l'événement `alarm` de la classe AlarmClock en appelant la méthode `addEventListener()`, héritée de EventDispatcher. Lorsqu'une occurrence de AlarmClock est prête à notifier à d'autres éléments de code le déclenchement de l'événement `alarm`, elle le fait en appelant la méthode `dispatchEvent()`, également héritée de EventDispatcher.

```
var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
this.dispatchEvent(alarm);
```

Ces lignes de code sont extraites de la méthode `onAlarm()` de la classe AlarmClock (présentée plus haut dans son intégralité). La méthode `dispatchEvent()` de l'occurrence de AlarmClock est appelée, puis elle notifie à tous les écouteurs enregistrés le déclenchement de l'événement `alarm` de l'occurrence de AlarmClock. Le paramètre transmis à `dispatchEvent()` est l'objet événement qui sera ensuite passé aux méthodes d'écouteur. Dans ce cas, il s'agit d'une occurrence de la classe AlarmEvent, une sous-classe de Event créée spécialement pour cet exemple.

Elaboration d'un événement d'alarme personnalisé

Tous les écouteurs d'événement reçoivent un paramètre d'objet événement avec des informations sur l'événement qui a été déclenché. Dans bien des cas, l'objet événement est une occurrence de la classe Event. Dans d'autres cas néanmoins, il s'avère utile de fournir des informations complémentaires aux écouteurs d'événement. Comme décrit plus haut dans ce chapitre, il suffit pour cela de définir une nouvelle classe, sous-classe de la classe Event, et d'utiliser une occurrence de cette classe comme objet événement. Dans cet exemple, une occurrence de AlarmEvent est utilisée comme objet événement lorsque l'événement `alarm` de la classe AlarmClock est distribué. La classe AlarmEvent, présentée ici, fournit des informations complémentaires sur l'événement `alarm`, à savoir le message d'alarme :

```
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;
    }
    ...
}
```

Le meilleur moyen de créer une classe d'objet événement personnalisée est de définir une classe qui étend la classEvent, comme illustré dans l'exemple précédent. Pour compléter la fonctionnalité héritée, la classe AlarmEvent définit une propriété `message` qui contient le texte du message d'alarme associé à l'événement. La valeur `message` est transmise sous forme de paramètre au constructeur AlarmEvent. La classe AlarmEvent définit également la constante `ALARM` qui peut servir à référencer l'événement (`alarm`) lors de l'appel de la méthode `addEventListener()` de la classe AlarmClock.

Outre l'ajout de fonctionnalité, chaque sous-classe Event doit redéfinir la méthode `clone()` héritée dans le cadre de la gestion des événements ActionScript. Les sous-classes Event peuvent éventuellement redéfinir la méthode `toString()` afin d'inclure les propriétés de l'événement personnalisé dans la valeur renvoyée par l'appel de la méthode `toString()`.

```
    /**
     * 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");
    }
```

La méthode `clone()` redéfinie doit renvoyer une nouvelle occurrence de la sous-classe Event personnalisée, avec toutes les propriétés personnalisées définies pour correspondre à l'occurrence actuelle. Dans la méthode `toString()` redéfinie, la méthode d'utilitaire `formatToString()` (héritée de Event) sert à fournir une chaîne comportant le nom du type personnalisé, ainsi que les noms et valeurs de toutes ses propriétés.

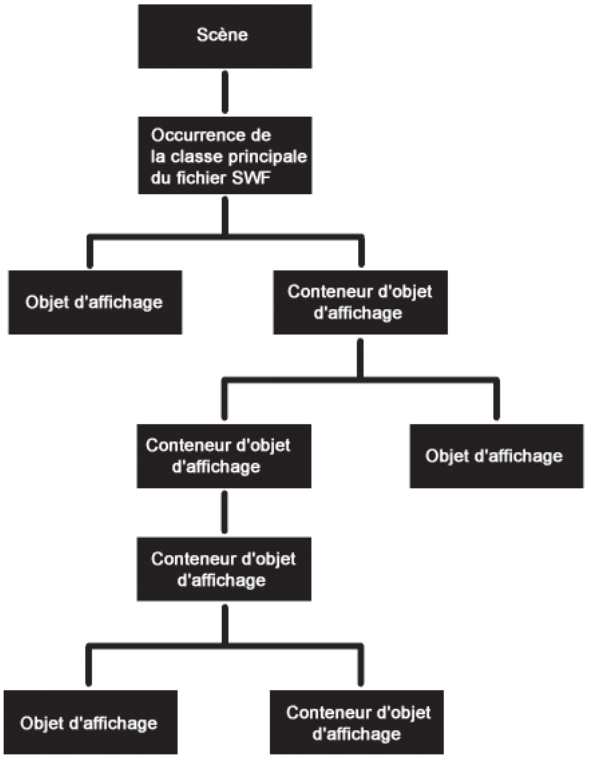
Chapitre 13 : Programmation de l'affichage

Dans Adobe® ActionScript® 3.0, la programmation de l'affichage vous permet de manipuler des éléments qui s'affichent sur la scène d'Adobe® Flash® Player ou Adobe® AIR™. Ce chapitre décrit les concepts fondamentaux de l'utilisation des éléments affichés à l'écran. Il décrit l'organisation par programmation des éléments visuels, ainsi que la création de classes d'objets d'affichage personnalisées.

Concepts fondamentaux de la programmation de l'affichage

Introduction à la programmation de l'affichage

Chaque application créée par le biais d'ActionScript 3.0 possède une hiérarchie d'objets d'affichage appelée *liste d'affichage*, comme l'indique l'illustration ci-dessous. La liste d'affichage contient tous les éléments visibles de l'application.



Comme le montre cette illustration, les éléments d'affichage se rangent dans un ou plusieurs groupes suivants :

- Scène

  La scène constitue le conteneur de base des objets d'affichage. Chaque application comporte un objet Stage, qui contient tous les objets d'affichage à l'écran. La scène correspond au conteneur de plus haut niveau et domine la hiérarchie de la liste d'affichage :

  Chaque fichier SWF est associé à une classe ActionScript, appelée *classe principale du fichier SWF*. Lorsqu'un fichier SWF s'ouvre dans Flash Player ou Adobe AIR, Flash Player ou AIR appelle la fonction constructeur correspondant à la classe et l'occurrence créée (systématiquement un type d'objet d'affichage) est ajoutée en tant qu'enfant de l'objet Stage. La classe principale d'un fichier SWF étend systématiquement la classe Sprite (pour plus d'informations, consultez la section « Avantages de l'utilisation de la liste d'affichage » à la page 282).

  Vous pouvez accéder à la scène via la propriété `stage` de toute occurrence de DisplayObject. Pour plus d'informations, consultez la section « Définition des propriétés de la scène » à la page 291.

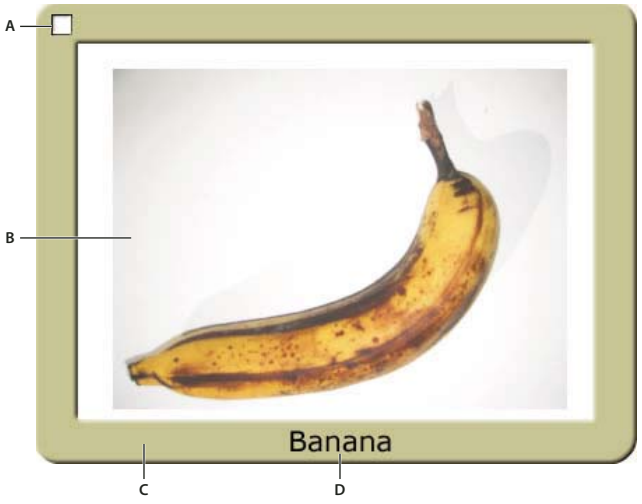
- Objets d'affichage

  Dans ActionScript 3.0, tous les éléments qui apparaissent à l'écran dans une application sont des types d'*objets d'affichage*. Le package flash.display comprend une classe DisplayObject, qui correspond à une classe de base étendue par diverses autres classes. Ces autres classes représentent divers types d'objets d'affichage, tels que les formes vectorielles, les clips et les champs de texte, pour n'en citer que quelques-uns. Pour une présentation de ces classes, consultez la section « Avantages de l'utilisation de la liste d'affichage » à la page 282.

- Conteneurs d'objets d'affichage

  Les conteneurs d'objets d'affichage sont des types spéciaux d'objets d'affichage qui, outre leur propre représentation visuelle, peuvent également comporter des objets enfant qui sont aussi des objets d'affichage.

  La classe DisplayObjectContainer est une sous-classe de la classe DisplayObject. Un objet DisplayObjectContainer peut contenir plusieurs objets d'affichage dans la *liste d'enfants* correspondante. Par exemple, l'illustration suivante contient un type d'objet DisplayObjectContainer appelé Sprite qui comporte divers objets d'affichage :



A. *Objet SimpleButton. Ce type d'objet d'affichage possède des états « up », « down » et « over ».* ***B.*** *Objet Bitmap. Dans ce cas de figure, l'objet Bitmap a été chargé à partir d'un JPEG externe via un objet Loader.* ***C.*** *Objet Shape. Le « cadre d'image » contient un rectangle arrondi dessiné dans ActionScript. Un filtre Ombre portée est appliqué à cet objet Shape.* ***D.*** *Objet TextField.*

Dans le contexte des objets d'affichage, les objets DisplayObjectContainer portent également le nom de *conteneurs d'objets d'affichage* voire, tout simplement, de *conteneurs*. Comme indiqué précédemment, la scène est un conteneur d'objets d'affichage.

Bien que tous les objets d'affichage visibles héritent leurs caractéristiques de la classe DisplayObject, le type de chacun d'eux correspond à une sous-classe déterminée de la classe DisplayObject. Il existe, par exemple, une fonction constructeur associée à la classe Shape ou à la classe Video, mais aucune fonction constructeur pour la classe DisplayObject.

Tâches courantes de programmation de l'affichage

Puisque la majeure partie de la programmation ActionScript implique de créer et manipuler des éléments visuels, un grand nombre de tâches se rapportent à la programmation de l'affichage. Ce chapitre décrit les tâches communes relatives à tous les objets d'affichage, notamment :

- Utilisation de la liste d'affichage et des conteneurs d'objets d'affichage
  - Ajout d'objets d'affichage à la liste d'affichage
  - Suppression d'objets de la liste d'affichage
  - Transfert d'objets entre les conteneurs d'objets d'affichage
  - Déplacement d'objets devant ou derrière d'autres objets
- Utilisation de la scène
  - Définition de la cadence
  - Contrôle de la mise à l'échelle de la scène
  - Utilisation du mode plein écran
- Manipulation des événements associés aux objets d'affichage
- Positionnement des objets d'affichage, notamment la création d'une interaction glisser-déposer
- Redimensionnement, mise à l'échelle et rotation d'un objet d'affichage
- Application de modes de fondu, de transformations de couleur et de transparence aux objets d'affichage
- Masquage des objets d'affichage
- Animation des objets d'affichage
- Chargement d'un contenu d'affichage externe (tel que des fichiers SWF ou des images)

Dans la suite de ce manuel, divers chapitres décrivent d'autres tâches de manipulation des objets d'affichage. Ces dernières concernent à la fois les tâches associées à tout objet d'affichage et les tâches réservées à des types déterminés d'objets d'affichage :

- Dessin de graphiques vectoriels par le biais d'ActionScript sur des objets d'affichage, décrit dans « Utilisation de l'API de dessin » à la page 328
- Application de transformations géométriques à des objets d'affichage, décrite dans « Utilisation de la géométrie » à la page 350
- Application d'effets de filtre graphique tels que le flou, le rayonnement, l'ombre portée, etc. à des objets d'affichage, décrite dans « Filtrage des objets d'affichage » à la page 363
- Utilisation de caractéristiques propres à MovieClip, décrite dans « Utilisation des clips » à la page 417
- Utilisation d'objets TextField, décrite dans « Utilisation de texte » à la page 443
- Utilisation des images bitmap, décrite dans « Utilisation des images bitmap » à la page 494

- Utilisation d'éléments vidéo, décrite dans « Utilisation de la vidéo » à la page 536

Concepts importants et terminologie

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Alpha : valeur colorimétrique représentant le montant de transparence (ou, plus précisément, le montant d'opacité) d'une couleur. Ainsi, une couleur dotée d'une valeur de canal alpha de 60 % n'affiche que 60 % de son intensité totale et est transparente à 40 %.
- Graphique bitmap : graphique défini en termes informatiques sous forme de grille (lignes et colonnes) de pixels de couleur. Les exemples courants de graphiques bitmap incluent les photos numériques et images similaires.
- Mode de mélange : indique l'interaction requise du contenu de deux images qui se chevauchent. En règle générale, une image opaque superposée à une autre image se contente de bloquer l'image placée sous elle, qui est donc totalement invisible. Toutefois, divers modes de mélange entraînent le mélange des couleurs de diverses façons de sorte que le résultat corresponde à une combinaison des deux images.
- Liste d'affichage : hiérarchie des objets d'affichage rendus sous forme de contenu visible à l'écran par Flash Player et AIR. La scène correspond à la racine de la liste d'affichage et tous les objets d'affichage associés à la scène ou à l'un de ses enfants composent la liste d'affichage (même si l'objet n'est pas à proprement parler rendu, parce qu'il réside en dehors de la scène, par exemple).
- Objet d'affichage : objet représentant un type de contenu visuel dans Flash Player ou AIR. La liste d'affichage ne contient que des objets d'affichage et toutes les classes d'objets d'affichage sont des sous-classes de la classe DisplayObject.
- Conteneur d'objet d'affichage : type spécial d'objet d'affichage qui, outre (généralement) sa propre représentation visuelle, peut comporter des objets d'affichage enfant.
- Classe principale du fichier SWF : classe qui définit le comportement de l'objet d'affichage de plus haut niveau d'un fichier SWF, soit, fondamentalement, la classe associée au fichier SWF en tant que tel. Ainsi, un fichier SWF créé dans l'outil de programmation Flash possède un « scénario principal » qui intègre tous les autres scénarios. La classe principale du fichier SWF correspond à la classe dont le scénario principal est une occurrence.
- Masquage : technique consistant à ne pas afficher certaines parties d'une image (ou, à l'inverse, à n'afficher que certaines parties d'une image). Les sections de l'image masque deviennent transparentes, afin d'assurer la visibilité du contenu sous-jacent. Ce terme se réfère à la bande utilisée par un peintre en bâtiment pour empêcher la peinture d'être appliquée à certaines sections.
- Scène : conteneur visuel correspondant à la base ou à l'arrière-plan de tout contenu visuel dans un fichier SWF.
- Transformation : modification des caractéristiques visuelles d'un graphique (rotation de l'objet, modification de son échelle, désalignement, déformation ou altération de sa couleur).
- Graphique vectoriel : graphique défini en termes informatiques par des lignes et des formes dessinées en fonction de caractéristiques déterminées (épaisseur, longueur, taille, angle et position, par exemple).

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester ses exemples de code. Ce chapitre étant consacré à la création et à la manipulation de contenu visuel, pratiquement tous les exemples de code qu'il contient créent des objets visuels et les affichent à l'écran. A l'encontre des chapitres précédents, tester un exemple de code implique de visualiser le résultat dans Flash Player ou AIR plutôt que d'afficher des valeurs de variable. Pour tester les codes de ce chapitre :

1. Créez un document vide à l'aide de l'outil de programmation Flash.

2 Sélectionnez une image-clé dans le scénario.

3 Ouvrez le panneau Actions et copiez le code dans le panneau Script.

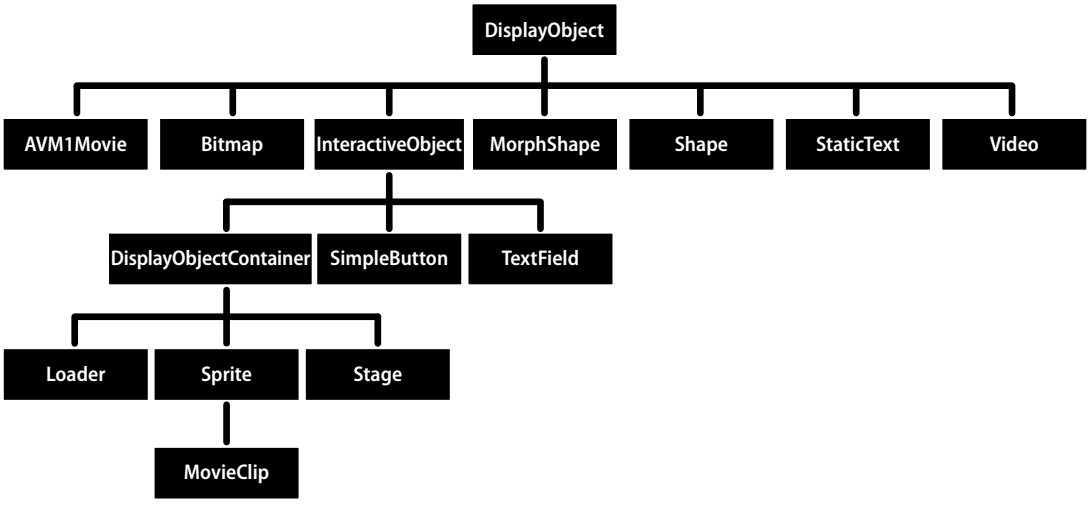
4 Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

Le résultat du code s'affiche à l'écran et tout appel de la fonction `trace()` apparaît dans le panneau Sortie.

Ces techniques de test d'exemples de code sont décrites de manière plus détaillée dans « Test des exemples de code contenus dans un chapitre » à la page 36.

Classes d'affichage de base

Le package flash.display ActionScript 3.0 contient des classes destinées aux objets visuels susceptibles d'apparaître dans Flash Player ou AIR. L'illustration suivante identifie les relations entre les sous-classes de ces classes d'objets d'affichage de base.



L'illustration indique ce dont héritent les classes d'objets d'affichage. Notez que certaines de ces classes, en particulier StaticText, TextField et Video, ne figurent pas dans le package flash.display, mais héritent toutefois des caractéristiques de la classe DisplayObject.

Toutes les classes qui étendent la classe DisplayObject héritent de ses méthodes et propriétés. Pour plus d'informations, consultez la section « Propriétés et méthodes de la classe DisplayObject » à la page 285.

Vous pouvez créer une occurrence d'un objet des classes suivantes, qui figurent dans le package flash.display :

- Bitmap : la classe Bitmap permet de définir des objets bitmap, qu'ils soient chargés à partir de fichiers externes ou rendus via ActionScript. Vous pouvez charger des bitmaps à partir de fichiers externes par le biais de la classe Loader. Libre à vous de charger des fichiers GIF, JPG ou PNG. Vous pouvez également créer un objet BitmapData à partir de données personnalisées, puis créer un objet Bitmap qui utilise ces données. Les méthodes de la classe BitmapData permettent de modifier les bitmaps, qu'ils soient chargés ou créés dans ActionScript. Pour plus d'informations, consultez la section « Chargement d'objets d'affichage » à la page 319 et le chapitre « Utilisation des images bitmap » à la page 494.
- Loader : la classe Loader permet de charger des ressources externes (fichiers SWF ou graphiques). Pour plus d'informations, consultez la section « Chargement dynamique du contenu d'affichage » à la page 319.

- Shape : la classe Shape permet de créer des graphiques vectoriels, tels que des rectangles, des lignes, des cercles, etc. Pour plus d'informations, consultez le chapitre « Utilisation de l'API de dessin » à la page 328.
- SimpleButton : un objet SimpleButton est une représentation ActionScript d'un symbole de bouton créé dans l'outil de programmation Flash. Une occurrence de SimpleButton est dotée de quatre états de bouton : « up », « down », « over » et « hit test » (zone qui réagit aux événements souris et clavier).
- Sprite : un objet Sprite peut contenir des graphiques qui lui sont propres, ainsi que des objets d'affichage enfant. (La classe Sprite étend la classe DisplayObjectContainer.) Pour plus d'informations, consultez la section « Utilisation de conteneurs d'objets d'affichage » à la page 286 et le chapitre « Utilisation de l'API de dessin » à la page 328.
- MovieClip : un objet MovieClip est la forme ActionScript d'un symbole de clip créé dans l'outil de programmation Flash. En pratique, un objet MovieClip est similaire à un objet Sprite, à une exception près : il possède également un scénario. Pour plus d'informations, consultez le chapitre « Utilisation des clips » à la page 417.

Les classes suivantes, qui ne figurent pas dans le package flash.display, sont des sous-classes de la classe DisplayObject :

- La classe TextField, qui figure dans le package flash.text, est un objet d'affichage destiné à l'affichage et à la saisie de texte. Pour plus d'informations, consultez la section « Utilisation de texte » à la page 443.
- La classe Video, qui figure dans le package flash.media, correspond à l'objet d'affichage utilisé pour afficher des fichiers vidéo. Pour plus d'informations, consultez le chapitre « Utilisation de la vidéo » à la page 536.

Les classes suivantes du package flash.display étendent la classe DisplayObject, mais il est impossible d'en créer une occurrence. Parce qu'elles combinent des fonctionnalités communes en une classe unique, elles servent plutôt de classes parent à d'autres objets d'affichage.

- AVM1Movie : la classe AVM1Movie permet de représenter des fichiers SWF chargés créés dans ActionScript 1.0 et 2.0.
- DisplayObjectContainer : les classes Loader, Stage, Sprite et MovieClip étendent chacune la classe DisplayObjectContainer. Pour plus d'informations, consultez la section « Utilisation de conteneurs d'objets d'affichage » à la page 286.
- InteractiveObject : classe de base de tous les objets utilisés pour interagir avec la souris et le clavier. Les objets SimpleButton, TextField, Loader, Sprite, Stage et MovieClip sont tous des sous-classes de la classe InteractiveObject. Pour plus d'informations sur la création d'interactions souris et clavier, consultez le chapitre « Capture des données saisies par l'utilisateur » à la page 608.
- MorphShape : ces objets sont générés lors de la création d'une interpolation de forme dans l'outil de programmation Flash. Il est impossible d'en créer des occurrences par le biais d'ActionScript, mais vous pouvez y accéder dans la liste d'affichage.
- Scène : la classe Stage étend la classe DisplayObjectContainer. Il n'existe qu'une seule occurrence de scène par application, et elle figure au sommet de la hiérarchie de la liste d'affichage. Vous pouvez accéder à la scène via la propriété `stage` de toute occurrence de DisplayObject. Pour plus d'informations, consultez la section « Définition des propriétés de la scène » à la page 291.

Par ailleurs, la classe StaticText, qui figure dans le package flash.text, étend la classe DisplayObject, mais il est impossible d'en créer une occurrence dans du code. Les champs de texte statique sont créés dans Flash uniquement.

Avantages de l'utilisation de la liste d'affichage

Dans ActionScript 3.0, des classes distinctes sont réservées aux différents types d'objets d'affichage. Dans ActionScript 1.0 et 2.0, un grand nombre de types d'objets identiques sont inclus dans une même classe : MovieClip.

Cette individualisation des classes et la structure hiérarchique des listes d'affichage présentent les avantages suivants :

- Rendu plus efficace et utilisation réduite de la mémoire
- Gestion optimisée de la profondeur
- Parcours entier de la liste d'affichage
- Objets d'affichage absents de la liste
- Classement simplifié en sous-classes des objets d'affichage

Rendu plus efficace et taille réduite des fichiers

Dans ActionScript 1.0 et 2.0, vous ne pouvez dessiner des formes que dans un objet MovieClip. ActionScript 3.0 intègre des classes d'objets d'affichage plus simples, dans lesquelles vous pouvez dessiner des formes. Parce que ces classes d'objets d'affichage ActionScript 3.0 ne contiennent pas le jeu complet de méthodes et propriétés associées à un objet MovieClip, elles mobilisent moins de ressources en mémoire et processeur.

Par exemple, à l'encontre d'un objet Shape, chaque objet MovieClip comporte des propriétés associées au scénario du clip. Les propriétés de gestion du scénario font parfois appel à un volume considérable de ressources en mémoire et processeur. Dans ActionScript 3.0, l'utilisation de l'objet Shape se traduit par une amélioration des performances. L'objet Shape nécessite moins de ressources que l'objet MovieClip, plus complexe. Flash Player et AIR n'ont pas besoin de gérer les propriétés MovieClip inutilisées, optimisant ainsi la vitesse et réduisant les besoins de mémoire de l'objet.

Gestion optimisée de la profondeur

Dans ActionScript 1.0 et 2.0, la profondeur était gérée par un système et des méthodes de gestion linéaire de la profondeur, tels que `getNextHighestDepth()`.

ActionScript 3.0 comprend la classe DisplayObjectContainer, dont les méthodes et propriétés sont mieux adaptées à la gestion de la profondeur des objets d'affichage.

Dans ActionScript 3.0, lorsque vous déplacez un objet d'affichage au sein de la liste des enfants d'une occurrence de DisplayObjectContainer, les autres enfants du conteneur d'objets d'affichage sont automatiquement repositionnés et des positions d'index enfant appropriées leur sont affectées dans le conteneur d'objets d'affichage.

Par ailleurs, ActionScript 3.0 permet systématiquement de détecter tous les objets enfant de tout conteneur d'objets d'affichage. Chaque occurrence de DisplayObjectContainer possède une propriété `numChildren`, qui indique le nombre d'enfants figurant dans le conteneur d'objets d'affichage. Puisque la liste des enfants d'un conteneur d'objets d'affichage correspond systématiquement à une liste indexée, vous pouvez examiner chaque objet de la liste de la position d'index 0 à la dernière position d'index (`numChildren - 1`). Cette technique n'était pas proposée par les méthodes et propriétés d'un objet MovieClip dans ActionScript 1.0 et 2.0.

ActionScript 3.0 permet de parcourir aisément et séquentiellement la liste d'affichage, car les numéros d'index de la liste des enfants d'un conteneur d'objets d'affichage se suivent. Parcourir la liste d'affichage et gérer la profondeur des objets est désormais beaucoup plus simple que dans ActionScript 1.0 et 2.0. Dans ActionScript 1.0 et 2.0, un clip pouvait en effet contenir des objets dont l'ordre de profondeur n'était pas séquentiel, ce qui rendait parfois le parcours de la liste d'objets difficile. Dans ActionScript 3.0, chaque liste d'enfants d'un conteneur d'objets d'affichage est mise en cache en interne sous forme de tableau, ce qui permet des recherches extrêmement rapides (par index). Passer en boucle sur tous les enfants d'un conteneur d'objets d'affichage s'avère également très rapide.

Dans ActionScript 3.0, vous pouvez également accéder aux enfants d'un conteneur d'objets d'affichage par le biais de la méthode `getChildByName()` de la classe DisplayObjectContainer.

Parcours entier de la liste d'affichage

ActionScript 1.0 et 2.0 ne vous permettaient pas d'accéder à certains objets, telles les formes vectorielles, dessinées dans l'outil de programmation Flash. Dans ActionScript 3.0, vous pouvez accéder à tous les objets de la liste d'affichage, qu'ils aient été créés en ActionScript ou dans l'outil de programmation Flash. Pour plus d'informations, consultez la section « Parcours de la liste d'affichage » à la page 290.

Objets d'affichage absents de la liste

ActionScript 3.0 permet de créer des objets d'affichage qui ne figurent pas dans la liste d'affichage visible. Ils portent le nom d'objets d'affichage *hors liste*. Un objet d'affichage n'est ajouté à la liste d'affichage visible que lorsque vous appelez la méthode `addChild()` ou `addChildAt()` d'une occurrence de DisplayObjectContainer qui a déjà été intégrée à la liste d'affichage.

Les objets d'affichage hors liste permettent d'assembler des objets d'affichage complexes, tels que ceux qui possèdent plusieurs conteneurs d'objets d'affichage comportant plusieurs objets d'affichage. En n'intégrant pas à la liste des objets d'affichage, vous pouvez assembler des objets complexes sans avoir à effectuer leur rendu. Vous économisez ainsi le temps de traitement correspondant. Vous pouvez alors ajouter un objet hors liste à la liste d'affichage au moment voulu. Il est également possible d'intégrer un enfant d'un conteneur d'objets d'affichage à la liste d'affichage, puis de l'en extraire ou d'en modifier la position dans cette dernière, le cas échéant.

Classement simplifié en sous-classes des objets d'affichage

Dans ActionScript 1.0 et 2.0, il était souvent nécessaire d'ajouter de nouveaux objets MovieClip à un fichier SWF pour créer des formes de base ou afficher des bitmaps. Dans ActionScript 3.0, la classe DisplayObject comprend un grand nombre de sous-classes intégrées, telles que Shape et Bitmap. Parce que les classes d'ActionScript 3.0 sont plus spécialisées pour des types spécifiques d'objets, il est plus simple de créer des sous-classes de base des classes intégrées.

Par exemple, pour dessiner un cercle dans ActionScript 2.0, vous pourriez créer une classe CustomCircle qui étend la classe MovieClip lors de la création d'une occurrence d'un objet de la classe personnalisée. Néanmoins, cette classe comprendrait également diverses propriétés et méthodes émanant de la classe MovieClip (telles que `totalFrames`) qui ne s'appliquent pas à elle. Dans ActionScript 3.0, vous pouvez toutefois créer une classe CustomCircle qui étend l'objet Shape et, de ce fait, ne comprend pas les propriétés et méthodes sans rapport contenues dans la classe MovieClip. Le code suivant illustre un exemple de classe 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);
    }
}
```

Utilisation des objets d'affichage

Maintenant que vous maîtrisez les bases de la scène, des objets d'affichage, des conteneurs d'objets d'affichage et de la liste d'affichage, cette section contient des informations plus détaillées relatives à l'utilisation des objets d'affichage dans ActionScript 3.0.

Propriétés et méthodes de la classe DisplayObject

Tous les objets d'affichage sont des sous-classes de la classe DisplayObject et, de ce fait, héritent des propriétés et méthodes de cette dernière. Les propriétés dont ils héritent correspondent aux propriétés de base qui s'appliquent à tous les objets d'affichage. Par exemple, chaque objet d'affichage possède une propriété `x` et une propriété `y` qui indiquent sa position dans son conteneur d'objets d'affichage.

Il est impossible de créer une occurrence de DisplayObject à l'aide du constructeur de la classe DisplayObject. Vous devez créer un autre type d'objet (un objet qui est une sous-classe de la classe DisplayObject), tel Sprite, pour créer une occurrence d'objet par le biais de l'opérateur `new`. Par ailleurs, pour créer une classe d'objet d'affichage personnalisée, vous devez créer une sous-classe de l'une des sous-classes d'objets d'affichage ayant une fonction constructeur utilisable (telle que la classe Shape ou la classe Sprite). Pour plus d'informations, consultez la description de la classe DisplayObject dans le Guide de référence du langage et des composants ActionScript 3.0.

Ajout d'objets d'affichage à la liste d'affichage

Lorsque vous créez une occurrence d'un objet d'affichage, elle n'apparaît pas à l'écran (sur la scène) tant que vous ne l'avez pas ajoutée à un conteneur d'objets d'affichage figurant dans la liste d'affichage. Par exemple, dans le code suivant, l'objet `myText` TextField n'est pas visible si vous omettez la dernière ligne de code. Dans la dernière ligne de code, le mot-clé `this` doit se référer à un conteneur d'objets d'affichage figurant déjà dans la liste d'affichage.

```
import flash.display.*;
import flash.text.TextField;
var myText:TextField = new TextField();
myText.text = "Buenos dias.";
this.addChild(myText);
```

Lorsque vous ajoutez un élément visuel à la scène, il devient un *enfant* de cette dernière. Le premier fichier SWF chargé dans une application (tel celui intégré à une page HTML) est automatiquement ajouté en tant qu'enfant de la scène. Il peut s'agir de n'importe quel type d'objet qui étend la classe Sprite.

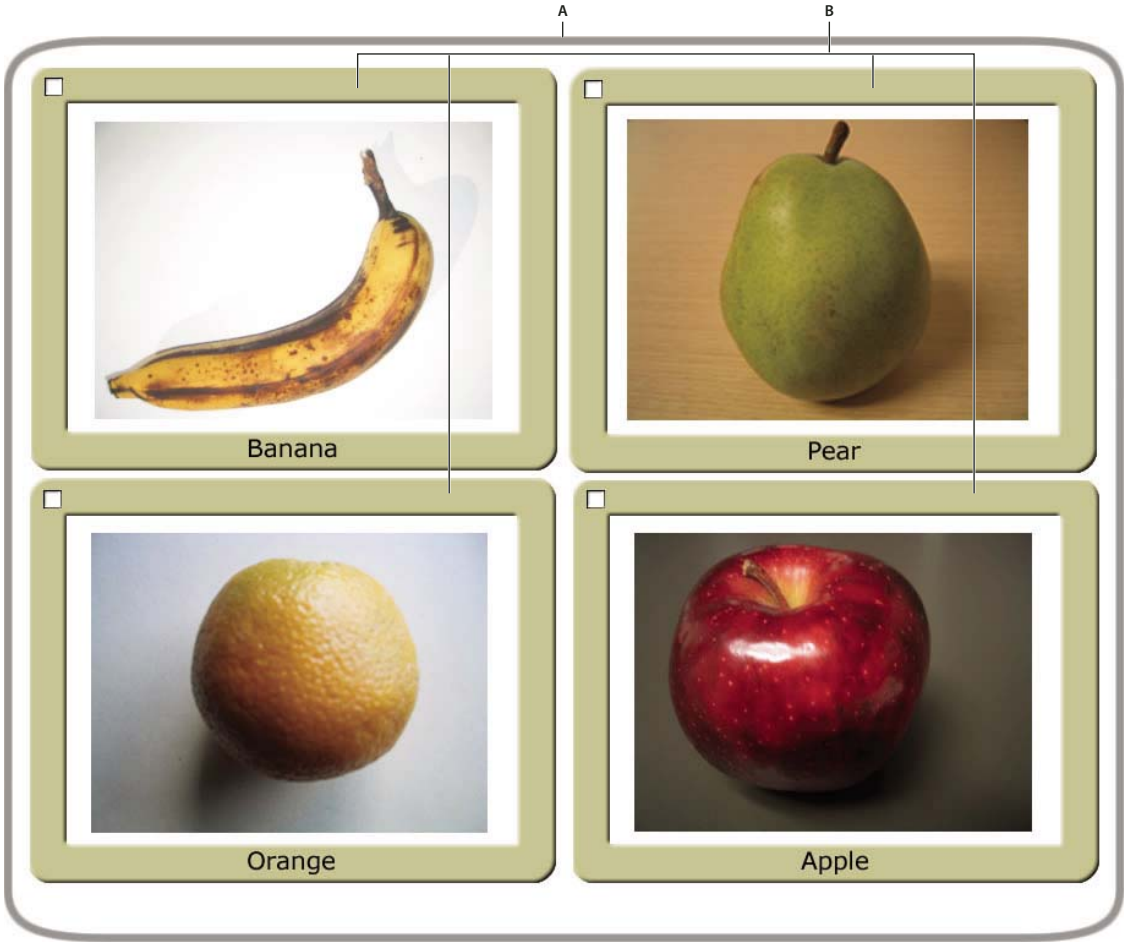
Tout objet d'affichage créé *sans* utiliser ActionScript (par exemple en ajoutant une balise MXML dans Adobe Flex Builder 3 ou en plaçant un élément sur la scène dans Flash) est ajouté à la liste d'affichage. Bien que vous n'ajoutiez pas ces objets d'affichage par le biais d'ActionScript, vous pouvez y accéder via ActionScript. Par exemple, le code suivant règle la largeur d'un objet appelé `button1`, qui a été ajouté dans l'outil de programmation (et non via ActionScript) :

```
button1.width = 200;
```

Utilisation de conteneurs d'objets d'affichage

Si un objet DisplayObjectContainer est supprimé de la liste d'affichage ou s'il est transféré ou transformé d'une autre façon, chaque objet d'affichage de DisplayObjectContainer est également supprimé, transféré ou transformé.

Un conteneur d'objets d'affichage correspond à un type d'objet d'affichage et peut être ajouté à un autre conteneur d'objets d'affichage. Par exemple, l'image suivante illustre un conteneur d'objets d'affichage, `pictureScreen`, qui comporte une forme de contour et quatre autres conteneurs d'objets d'affichage (de type PictureFrame) :



A. *Forme définissant la bordure du conteneur d'objets d'affichage pictureScreen* ***B.*** *Quatre conteneurs d'objets d'affichage, qui sont des enfants de l'objet pictureScreen*

Pour qu'un objet d'affichage apparaisse dans la liste d'affichage, vous devez l'ajouter à un conteneur d'objets d'affichage figurant dans la liste d'affichage. A cet effet, vous utilisez la méthode `addChild()` ou la méthode `addChildAt()` de l'objet conteneur. Par exemple, sans la dernière ligne du code suivant, l'objet `myTextField` ne s'afficherait pas :

```
var myTextField:TextField = new TextField();
myTextField.text = "hello";
this.root.addChild(myTextField);
```

Dans cet exemple de code, `this.root` pointe vers le conteneur d'objets d'affichage MovieClip qui comporte le code. Dans votre propre code, vous pouvez stipuler un autre conteneur.

Utilisez la méthode `addChildAt()` pour ajouter l'enfant à une position déterminée de la liste des enfants du conteneur d'objets d'affichage. Ces positions d'index basées sur zéro dans la liste des enfants se réfèrent à l'ordre d'apparition (de l'avant à l'arrière) des objets d'affichage. Considérons par exemple les trois objets d'affichage suivants. Chaque objet a été créé à partir d'une classe personnalisée appelée Ball.

L'ordre d'apparition de ces objets d'affichage dans leur conteneur peut être modifié par le biais de la méthode `addChildAt()`. Considérons par exemple le code qui suit :

```
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);
```

Une fois ce code exécuté, les objets d'affichage sont placés comme suit dans l'objet DisplayObjectContainer `container`. Notez l'ordre d'apparition des objets.

Pour placer un objet en tête de la liste d'affichage, il suffit de l'ajouter à nouveau à celle-ci. Par exemple, après le code précédent, utilisez la ligne de code suivante pour placer `ball_A` en première position dans la pile :

```
container.addChild(ball_A);
```

Ce code supprime `ball_A` de son emplacement actuel dans la liste d'affichage de `container`, et l'ajoute ensuite au sommet de la liste, ce qui a pour effet de le placer en haut de l'empilement d'objets.

Vous disposez de la méthode `getChildAt()` pour vérifier l'ordre d'apparition des objets d'affichage. La méthode `getChildAt()` renvoie les objets enfant d'un conteneur en fonction du numéro d'index transmis. Par exemple, le code suivant révèle le nom des objets d'affichage placés à des positions diverses dans la liste des enfants de l'objet DisplayObjectContainer `container` :

```
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_C
trace(container.getChildAt(2).name); // ball_B
```

Si vous supprimez un objet d'affichage de la liste des enfants de son conteneur parent, les éléments de la liste ayant un indice plus élevé descendent tous d'une position dans l'index des enfants. Ainsi, si nous reprenons l'exemple précédent, le code ci-après illustre le transfert de l'objet d'affichage qui occupait la position 2 dans l'objet DisplayObjectContainer `container` vers la position 1 suite à la suppression d'un objet d'affichage occupant une position inférieure dans la liste d'enfants :

```
container.removeChild(ball_C);
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_B
```

Les méthodes `removeChild()` et `removeChildAt()` ne suppriment pas entièrement une occurrence d'objet d'affichage. Elles se contentent de la supprimer de la liste des enfants du conteneur. Une autre variable peut continuer à faire référence à l'occurrence. (Utilisez l'opérateur `delete` pour supprimer totalement un objet.)

Un objet d'affichage ne possédant qu'un seul conteneur parent, vous ne pouvez ajouter une occurrence d'objet d'affichage qu'à un seul conteneur d'objets d'affichage. Par exemple, le code suivant indique que l'objet d'affichage `tf1` ne peut figurer que dans un seul conteneur (soit, dans ce cas, un Sprite, qui étend la classe 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
```

Si vous ajoutez à un conteneur d'objets d'affichage un objet qui est déjà contenu dans un autre conteneur d'objets d'affichage, l'objet sera supprimé de la liste des enfants de ce dernier.

Outre les méthodes décrites précédemment, la classe DisplayObjectContainer définit plusieurs méthodes d'utilisation des objets d'affichage enfant, notamment :

- `contains()` : détermine si un objet d'affichage est un enfant d'un objet DisplayObjectContainer.
- `getChildByName()` : extrait un objet d'affichage en fonction de son nom.
- `getChildIndex()` : renvoie la position d'index d'un objet d'affichage.
- `setChildIndex()` : modifie la position d'un objet d'affichage enfant.
- `swapChildren()` : permute l'ordre de deux objets d'affichage.

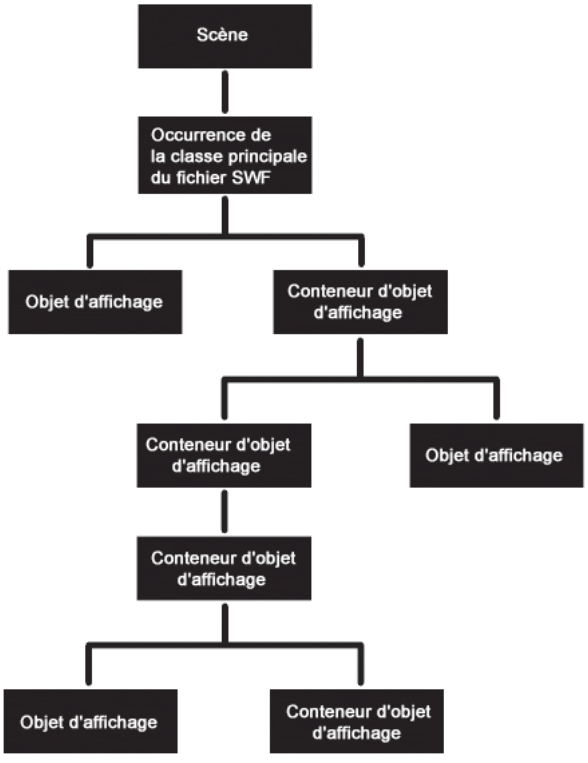
- `swapChildrenAt()` : permute l'ordre de deux objets d'affichage définis en fonction de leur valeur d'index.

Pour plus d'informations, consultez les entrées appropriées du Guide de référence du langage et des composants ActionScript 3.0.

N'oubliez pas qu'un objet d'affichage qui ne figure pas dans la liste d'affichage (donc, qui ne se trouve pas dans un conteneur d'objets d'affichage enfant de la scène) est appelé objet d'affichage *hors liste*.

Parcours de la liste d'affichage

Comme nous l'avons vu, la liste d'affichage est une structure en arborescence. Au sommet de l'arborescence figure la scène, qui peut comporter plusieurs objets d'affichage. Les objets d'affichage qui sont eux-mêmes des conteneurs d'objets d'affichage peuvent contenir d'autres objets d'affichage, voire des conteneurs d'objets d'affichage.



La classe DisplayObjectContainer comporte des propriétés et méthodes de parcours de la liste d'affichage, par le biais des listes d'enfants des conteneurs d'objets d'affichage. Considérons par exemple le code suivant, qui ajoute deux objets d'affichage, `title` et `pict`, à l'objet `container` (qui est un Sprite, et la classe Sprite étend la classe 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);
```

La méthode `getChildAt()` renvoie l'enfant de la liste d'affichage à une position d'index déterminée :

```
trace(container.getChildAt(0) is TextField); // true
```

Vous pouvez également accéder aux objets enfant en indiquant leur nom. Chaque objet d'affichage possède un nom, qui est attribué par défaut par Flash Player ou AIR (tel « `instance1` ») si vous ne l'attribuez pas vous-même. Par exemple, le code suivant indique comment utiliser la méthode `getChildByName()` pour accéder à un objet d'affichage enfant portant le nom « `banana loader` » :

```
trace(container.getChildByName("banana loader") is Loader); // true
```

L'utilisation de la méthode `getChildByName()` entraîne parfois un ralentissement des performances par rapport à la méthode `getChildAt()`.

Puisque la liste d'affichage d'un conteneur d'objets d'affichage peut contenir d'autres conteneurs d'objets d'affichage en tant qu'objets enfant, vous pouvez parcourir la liste d'affichage complète de l'application sous forme d'arborescence. Par exemple, dans l'extrait de code illustré précédemment, une fois l'opération de chargement de l'objet Loader `pict` terminée, un objet d'affichage enfant (l'image bitmap) de l'objet `pict` est chargé. Pour accéder à cet objet d'affichage bitmap, vous pouvez écrire `pict.getChildAt(0)`. Vous pouvez également écrire `container.getChildAt(0).getChildAt(0)` (puisque `container.getChildAt(0) == pict`).

La fonction suivante génère un extrait `trace()` en retrait de la liste d'affichage d'un conteneur d'objets d'affichage :

```
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 + "")
        }
    }
}
```

Définition des propriétés de la scène

La classe Stage annule la plupart des propriétés et méthodes de la classe DisplayObject. Si vous appelez l'une de ces propriétés ou méthodes annulées, Flash Player et AIR renvoient une exception. Par exemple, l'objet Stage ne possède pas de propriété `x` ou `y` car, en tant que conteneur principal de l'application, sa position est fixe. Or, les propriétés `x` et `y` indiquent la position d'un objet d'affichage par rapport à son conteneur, et puisque la scène ne se trouve pas dans un autre conteneur d'objets d'affichage, ces propriétés seraient inapplicables.

Remarque : *certaines propriétés et méthodes de la classe Stage sont réservées aux objets d'affichage appartenant au même sandbox de sécurité que le premier fichier SWF chargé. Pour plus d'informations, consultez la section « Sécurité de la scène » à la page 732.*

Contrôle de la cadence de lecture

La propriété `framerate` de la classe Stage permet de définir la cadence de tous les fichiers SWF chargés dans l'application. Pour plus d'informations, consultez le Guide de référence du langage et des composants ActionScript 3.0.

Contrôle de la mise à l'échelle de la scène

Lorsque la portion de l'écran qui représente Flash Player ou AIR est redimensionnée, Flash Player ou AIR ajuste automatiquement le contenu de la scène pour compenser. La propriété `scaleMode` de la classe Stage détermine comment le contenu de la scène est ajusté. Cette propriété peut être réglée sur quatre valeurs différentes, définies en tant que constantes dans la classe flash.display.StageScaleMode.

Pour trois des valeurs `scaleMode` (`StageScaleMode.EXACT_FIT`, `StageScaleMode.SHOW_ALL` et `StageScaleMode.NO_BORDER)`, Flash Player et AIR mettent à l'échelle le contenu de la scène en fonction de ses limites. Les trois options déterminent différemment l'exécution de la mise à l'échelle :

- `StageScaleMode.EXACT_FIT` applique une mise à l'échelle proportionnelle du fichier SWF.
- `StageScaleMode.SHOW_ALL` détermine si une bordure apparaît, telles les barres noires qui s'affichent lorsque vous regardez un film grand écran sur une télévision standard.
- `StageScaleMode.NO_BORDER` détermines'il est possible de réduire partiellement le contenu.

Si `scaleMode` est défini sur `StageScaleMode.NO_SCALE`, le contenu de la scène conserve sa taille stipulée lorsque l'utilisateur redimensionne la fenêtre Flash Player ou AIR. Dans ce mode de mise à l'échelle uniquement, les propriétés `stageWidth` et `stageHeight` de la classe Stage permettent de déterminer les dimensions réelles de la fenêtre redimensionnée, en pixels. (Dans les autres modes, les propriétés `stageWidth` et `stageHeight` renvoient toujours la largeur et la hauteur originales du fichier SWF.) De plus, lorsque la propriété `scaleMode` est définie sur `StageScaleMode.NO_SCALE` et que le fichier SWF est redimensionné, l'événement `resize` de la classe Stage est distribué, ce qui vous permet d'effectuer des ajustements en conséquence.

Définir `scaleMode` sur `StageScaleMode.NO_SCALE` permet donc de mieux contrôler l'ajustement du contenu en cas de redimensionnement de la fenêtre. Par exemple, si un fichier SWF contient une vidéo et une barre de contrôle, il peut s'avérer utile de conserver la taille de la barre de contrôle en cas de redimensionnement de la scène et de ne modifier que la taille de la fenêtre de vidéo en fonction du changement de taille de la scène. Ce cas de figure est illustré dans l'exemple suivant :

```
// videoScreen is a display object (e.g. a Video instance) containing a
// video; 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 videoScreen) 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 = videoScreen.stage;
swfStage.scaleMode = StageScaleMode.NO_SCALE;
swfStage.align = StageAlign.TOP_LEFT;

function resizeDisplay(event:Event):void
{
    var swfWidth:int = swfStage.stageWidth;
    var swfHeight:int = swfStage.stageHeight;

    // Resize the video window.
    var newVideoHeight:Number = swfHeight - controlBar.height;
    videoScreen.height = newVideoHeight;
    videoScreen.scaleX = videoScreen.scaleY;

    // Reposition the control bar.
    controlBar.y = newVideoHeight;
}

swfStage.addEventListener(Event.RESIZE, resizeDisplay);
```

Utilisation du mode plein écran

Le mode plein écran permet de définir la scène d'un film de sorte à remplir totalement le moniteur sans bordure ou menu. La propriété `displayState` de la classe Stage permet d'activer ou désactiver ce mode pour un fichier SWF. La propriété `displayState` peut être réglée sur l'une des valeurs définies par les constantes de la classe flash.display.StageDisplayState. Pour activer le mode plein écran, définissez la propriété `displayState` sur `StageDisplayState.FULL_SCREEN`:

```
stage.displayState = StageDisplayState.FULL_SCREEN;
```

Dans Flash Player, le mode plein écran ne peut être activé que via ActionScript en réponse à un clic de souris (clic de bouton droit inclus) ou une frappe de touche. Le contenu AIR qui s'exécute dans le sandbox de sécurité de l'application ne nécessite pas que le mode plein écran soit activé en réponse à une action de l'utilisateur.

Pour quitter le mode plein écran, définissez la propriété `displayState` sur `StageDisplayState.NORMAL`.

```
stage.displayState = StageDisplayState.NORMAL;
```

Un utilisateur peut également désactiver le mode plein écran en plaçant le focus sur une autre fenêtre ou en utilisant l'une des combinaisons de touches suivantes : la touche Echap (toutes les plates-formes), Contrôle-W (Windows), Commande-W (Mac) ou Alt-F4 (Windows).

Activation du mode plein écran dans Flash Player

Pour activer le mode plein écran d'un fichier SWF intégré à une page HTML, le code HTML requis pour intégrer Flash Player doit comprendre une balise `param` et un attribut `embed`, associés au nom `allowFullScreen` et à la valeur `true`, comme suit :

```
<object>
    ...
    <param name="allowFullScreen" value="true" />
    <embed ... allowfullscreen="true" />
</object>
```

Dans l'outil de programmation Flash, choisissez Fichier -> Paramètres de publication, puis dans la boîte de dialogue Paramètres de publication, cliquez sur l'onglet HTML et sélectionnez le modèle Flash seulement - Autorisation du Plein écran.

Dans Flex, assurez-vous que le modèle HTML inclut les balises `<object>` et `<embed>` qui prennent en charge le plein écran.

Si vous utilisez JavaScript dans une page Web pour générer les balises d'intégration de SWF, vous devez modifier le code JavaScript pour ajouter la balise et l'attribut `allowFullScreen param`. Par exemple, si votre page HTML fait appel à la fonction `AC_FL_RunContent()` (utilisée par les pages HTML générées par Flex Builder et Flash), vous devez ajouter le paramètre `allowFullScreen` à cet appel de fonction, comme suit :

```
AC_FL_RunContent(
    ...
    'allowFullScreen','true',
    ...
    ); //end AC code
```

Ce cas de figure ne s'applique pas aux fichiers SWF qui s'exécutent dans la version autonome de Flash Player.

Remarque : *si vous définissez le Mode fenêtre (wmode dans le code HTML) sur Opaque sans fenêtre (opaque) ou Transparent sans fenêtre (transparent), la fenêtre plein écran est toujours opaque.*

Il existe également des restrictions liées à la sécurité lors de l'utilisation du mode plein écran avec Flash Player dans un navigateur. Ces restrictions sont décrites dans le chapitre « Sécurité dans Flash Player » à la page 714.

Mise à l'échelle et taille de la scène en mode plein écran

Les propriétés `Stage.fullScreenHeight` et `Stage.fullScreenWidth` renvoient la hauteur et la largeur de l'écran utilisé lors de l'activation du mode plein écran, lorsque le passage à cet état est immédiat. Ces valeurs risquent d'être incorrectes si l'utilisateur déplace le navigateur d'un écran à un autre après avoir récupéré ces valeurs, mais avant de passer au mode plein écran. Si l'utilisateur récupère ces valeurs dans le gestionnaire d'événement dans lequel il a défini la propriété `Stage.displayState` sur `StageDisplayState.FULL_SCREEN`, celles-ci sont correctes. Pour les utilisateurs qui utilisent plusieurs écrans, le contenu du fichier SWF s'étend pour n'occuper qu'un seul écran. Flash Player et AIR utilisent une mesure pour identifier le moniteur contenant la portion la plus élevée du fichier SWF et activent le mode plein écran sur celui-ci. Les propriétés fullScreenHeight et fullScreenWidth indiquent uniquement la taille de l'écran utilisée pour le mode plein écran. Pour plus d'informations, consultez `Stage.fullScreenHeight` et `Stage.fullScreenWidth` dans le Guide de référence du langage et des composants ActionScript 3.0.

En mode plein écran, le comportement de mise à l'échelle de la scène est identique à celui du mode normal. Cette mise à l'échelle est contrôlée par la propriété `scaleMode` de la classe Stage. Si la propriété `scaleMode` est définie sur `StageScaleMode.NO_SCALE`, les propriétés `stageWidth` et `stageHeight` de la classe Stage sont modifiées pour répercuter la taille de la zone de l'écran occupée par le fichier SWF (soit, dans ce cas, l'écran entier). Dans un navigateur, le paramètre HTML correspondant contrôle le réglage.

L'événement `fullScreen` de la classe Stage permet de détecter et répondre à l'activation ou à la désactivation du mode plein écran. Par exemple, il peut être nécessaire de repositionner, d'ajouter ou de supprimer des éléments lors d'un changement d'état du mode plein écran, comme illustré par cet exemple :

```
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);
```

Comme l'indique ce code, l'objet associé à l'événement `fullScreen` est une occurrence de la classe flash.events.FullScreenEvent, dont la propriété `fullScreen` indique si le mode plein écran est activé (`true`) ou non (`false`).

Prise en charge du clavier en mode plein écran

Lorsque Flash Player est exécuté dans un navigateur, l'ensemble du code ActionScript lié au clavier (événements de clavier et saisie de texte dans des occurrences de TextField, entre autres), est désactivé en mode plein écran. Les exceptions (c'est-à-dire les touches qui restent actives) sont les suivantes :

- Les touches hors impression sélectionnées, notamment les touches fléchées, la barre d'espacement et la touche de tabulation
- Les raccourcis clavier qui désactivent le mode plein écran : touche Echap (Windows et Mac), Contrôle-W (Windows), Commande-W (Mac) et Alt-F4

Ces restrictions ne sont pas présentes pour le contenu SWF s'exécutant dans l'application Flash Player autonome ou dans AIR. AIR prend en charge un mode plein écran interactif qui permet la saisie clavier.

Mise à l'échelle matérielle en mode plein écran

La propriété `fullScreenSourceRect` de la classe Stage permet d'utiliser Flash Player ou AIR pour mettre à l'échelle une zone déterminée de la scène en mode plein écran. Dans Flash Player et AIR, la mise à l'échelle matérielle, si elle est disponible, s'effectue à l'aide de la carte graphique et de la carte vidéo de l'ordinateur, et permet généralement d'afficher le contenu plus rapidement que la mise à l'échelle logicielle.

Pour tirer parti de la mise à l'échelle matérielle, définissez l'ensemble ou une partie de la scène sur le mode plein écran. Le code suivant ActionScript 3.0 définit l'ensemble de la scène en mode plein écran :

```
import flash.geom.*;
{
    stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
    stage.displayState = StageDisplayState.FULL_SCREEN;
}
```

Lorsque cette propriété est définie sur un rectangle valide et la propriété `displayState` sur le mode plein écran, Flash Player et AIR redimensionnent la zone spécifiée. La taille réelle de la scène en pixels dans ActionScript ne change pas. Flash Player et AIR imposent une taille limite au rectangle en fonction de la taille du message standard « Appuyez sur la touche Echap pour quitter le mode plein écran ». Cette limite est généralement d'environ 260 sur 30 pixels, mais peut varier en fonction de la plate-forme et de la version de Flash Player.

La propriété `fullScreenSourceRect` ne peut être définie que lorsque Flash Player ou AIR ne sont pas en mode plein écran. Pour utiliser correctement cette propriété, vous devez tout d'abord la définir, puis définir la propriété `displayState` sur le mode plein écran.

Pour activer la mise à l'échelle, définissez la propriété `fullScreenSourceRect` sur un objet rectangle.

```
stage.fullScreenSourceRect = new Rectangle(0,0,320,240);
```

Pour désactiver la mise à l'échelle, définissez la propriété `fullScreenSourceRect` sur `null`.

```
stage.fullScreenSourceRect = null;
```

Pour tirer parti de toutes les fonctions d'accélération matérielle avec Flash Player, activez cette fonction dans la boîte de dialogue des paramètres de Flash Player. Pour afficher cette boîte de dialogue, cliquez avec le bouton droit de la souris (Windows) ou cliquez tout en appuyant sur la touche Ctrl (Mac) dans le contenu Flash Player dans votre navigateur. Cliquez sur le premier onglet, Affichage, puis cochez la case Activer l'accélération matérielle.

Modes Fenêtre direct et composition GPU

Flash Player 10 propose deux modes de fenêtre, direct et composition GPU, que vous pouvez activer dans les paramètres de publication de l'outil de programmation Flash. Ces modes ne sont pas pris en charge par AIR. Pour tirer parti de ces modes, vous devez activer l'accélération matérielle pour Flash Player.

Le mode direct utilise le chemin le plus rapide et le plus direct pour placer des graphismes sur l'écran, ce qui est pratique pour la lecture vidéo.

La composition GPU utilise l'unité de traitement graphique sur la carte vidéo pour accélérer la composition. La composition vidéo consiste à créer des images multi-couches pour aboutir à une seule image vidéo. Lorsque la composition est accélérée par la GPU, les performances de la conversion YUV, la correction des couleurs, la rotation, le redimensionnement et le fondu sont nettement améliorés. La conversion YUV se réfère à la conversion des couleurs de signaux analogiques composites, qui sont utilisées pour la transmission, au modèle de couleurs RVB (rouge-vert-bleu) que les caméras vidéo et les écrans utilisent. Le recours à la GPU pour accélérer la composition réduit la demande en mémoire et en puissance de calcul qui affectera les performances de l'unité centrale. Il en résulte une lecture vidéo plus régulière en définition standard.

Soyez vigilant lorsque vous implémentez ces modes Fenêtre. Fenêtre L'utilisation de la composition GPU peut s'avérer coûteuse en mémoire et en ressources de l'unité centrale. Si certaines opérations (comme les modes fondu, le filtrage, l'écrêtage ou le masquage) ne peuvent pas être exécutées dans la GPU, c'est le logiciel qui s'en charge. Adobe vous recommande de vous limiter à un fichier SWF par page HTML lorsque vous utilisez ces modes. Il ne faudrait pas les utiliser pour des bandeaux. La Flash Test Movie facility ne fait pas appel à l'accélération matérielle mais vous pouvez l'utiliser par le biais de l'option Aperçu avant publication.

Il est inutile de fixer une cadence supérieure à 60 (c'est la fréquence de régénération d'écran maximale) dans votre fichier SWF. Une cadence entre 50 et 55 débouche sur des images perdues, ce qui peut se produire de temps à autre pour des raisons diverses.

Le mode direct requiert Microsoft DirectX 9 avec une mémoire virtuelle RAM de 128 Mo sous Windows et OpenGL pour Apple Macintosh, Mac OS X version 10.2 ou ultérieure. La composition GPU requiert Microsoft DirectX 9 et la prise en charge de Pixel Shader 2.0 sous Windows avec une mémoire virtuelle RAM de 128 Mo. Sous Mac OS X et Linux, la composition GPU requiert OpenGL 1.5 et plusieurs extensions d'OpenGL (objet framebuffer, objets multitexture et shader, langage d'ombrage, shader de fragments).

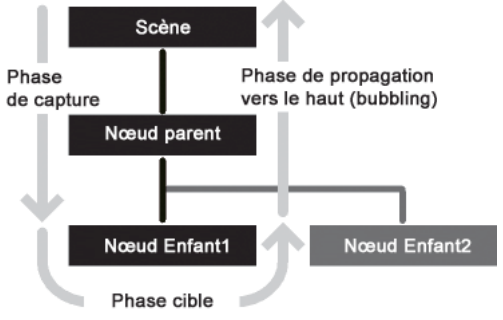
Vous pouvez activer les modes d'accélération `direct` et `gpu` pour chaque SWF par le biais de la boîte de dialogue Paramètres de publication de Flash, à l'aide du menu d'accélération matérielle sur l'onglet Flash. Si vous choisissez Aucun, le mode Fenêtre revient à `défaut`, `transparent` ou `opaque`, selon le paramètre du Mode Fenêtre de l'onglet HTML.

Manipulation des événements associés aux objets d'affichage

La classe DisplayObject hérite de la classe EventDispatcher. Cela signifie que chaque objet d'affichage peut être pleinement impliqué dans le modèle d'événement (décrit dans le chapitre « Gestion des événements » à la page 254). Chaque objet d'affichage peut utiliser sa méthode `addEventListener()` (héritée de la classe EventDispatcher) pour attendre un événement particulier, mais ceci uniquement si l'objet écouteur fait partie du flux d'événement de l'événement considéré.

Lorsque Flash Player ou AIR distribue un objet événement, celui-ci effectue un aller-retour à partir de la scène via l'objet d'affichage où s'est produit l'événement. Par exemple, si un utilisateur clique sur un objet d'affichage appelé `child1`, Flash Player distribue un objet événement à partir de la scène via la hiérarchie de la liste d'affichage jusqu'à l'objet d'affichage `child1`.

D'un point de vue conceptuel, le flux d'événement est divisé en trois phases, comme illustré par le diagramme suivant :



Pour plus d'informations, consultez le chapitre « Gestion des événements » à la page 254.

Lors de la gestion des événements liés aux objets d'affichage, il est important de ne pas oublier l'effet potentiel des objets écouteurs d'événement sur l'éventuelle suppression automatique des objets d'affichage de la mémoire (garbage collection) lorsqu'ils sont supprimés de la liste d'affichage. Si des objets d'un objet d'affichage sont enregistrés en tant qu'écouteurs d'événement, ce dernier n'est pas supprimé de la mémoire même s'il est supprimé de la liste d'affichage, car il continue à posséder des références à ces objets écouteur. Pour plus d'informations, consultez la section « Gestion des écouteurs d'événement » à la page 268.

Sélection d'une sous-classe de DisplayObject

Plusieurs options étant disponibles, l'une des décisions importantes à prendre lors de l'utilisation d'objets d'affichage est le choix de l'objet à utiliser dans un but précis. Les directives suivantes vous aideront à choisir l'option appropriée. Ces suggestions sont valides que vous créiez une occurrence de classe ou que vous choisissiez une classe de base pour une classe en cours de création :

- Si vous n'avez pas besoin d'un objet pouvant contenir d'autres objets d'affichage (autrement dit, si vous avez simplement besoin d'un objet à utiliser comme élément isolé), choisissez l'une des sous-classes suivantes de DisplayObject ou d'InteractiveObject, selon l'utilisation prévue :
  - Bitmap, pour afficher une image bitmap.
  - TextField, pour ajouter du texte.
  - Video, pour afficher une vidéo.
  - Shape, pour disposer d'une « toile » destinée au tracé d'un contenu à l'écran. En particulier si vous souhaitez créer une occurrence pour dessiner des formes à l'écran et qu'elle ne contient pas d'autres objets d'affichage, vous obtiendrez des performances nettement supérieures en utilisant Shape plutôt que Sprite ou MovieClip.
  - MorphShape, StaticText ou SimpleButton pour des éléments créés par l'outil de programmation Flash. Il est impossible de créer par programmation des occurrences de ces classes, mais vous pouvez créer des variables avec ces types de données pour pointer sur des éléments créés dans l'outil de programmation Flash.
- Si vous devez disposer d'une variable qui se réfère à la scène principale, utilisez la classe Stage en tant que type de données.
- Si vous devez disposer d'un conteneur pour charger un fichier SWF ou fichier image externe, utilisez une occurrence de Loader. Le contenu chargé sera ajouté à la liste d'affichage en tant qu'enfant de l'occurrence de Loader. Son type de données variera selon la nature du contenu chargé, comme suit :
  - Une image chargée est une occurrence de Bitmap.
  - Un fichier SWF chargé écrit dans ActionScript 3.0 est une occurrence de Sprite ou MovieClip (ou une occurrence d'une sous-classe de ces classes, comme indiqué par le créateur du contenu).
  - Un fichier SWF chargé écrit dans ActionScript 1.0 ou ActionScript 2.0 est une occurrence d'AVM1Movie.
- Si vous avez besoin d'un objet qui sera le conteneur d'autres objets d'affichage (que vous ayez ou non l'intention de dessiner en ActionScript dans cet objet), choisissez l'une des sous-classes de DisplayObjectContainer :
  - Sprite, si l'objet est créé en ActionScript uniquement ou en tant que classe de base d'un objet d'affichage personnalisé créé et manipulé en ActionScript uniquement.
  - MovieClip, si vous créez une variable pointant vers un symbole de clip créé dans l'outil de programmation Flash.
- Si vous créez une classe qui sera associée à un symbole de clip de la bibliothèque de Flash, choisissez l'une des sous-classes de DisplayObjectContainer comme classe de base de votre future classe :
  - MovieClip, si le symbole de clip associé possède un contenu sur plusieurs images.
  - Sprite, si le symbole de clip associé ne possède un contenu que sur la première image.

Manipulation des objets d'affichage

Quel que soit l'objet d'affichage utilisé, tous les éléments affichés à l'écran partagent diverses techniques de manipulations. Par exemple, ils peuvent tous être positionnés à l'écran, avancer ou reculer dans l'ordre d'empilement des objets d'affichage, mis à l'échelle, pivotés, et ainsi de suite. Dans la mesure où tous les objets d'affichage héritent ces fonctionnalités de leur classe de base commune (DisplayObject), lesdites fonctionnalités ont le même comportement pour une occurrence d'un objet TextField, Video, Shape ou tout autre objet d'affichage. Les sections suivantes passent en revue plusieurs de ces manipulations appliquées à tous les objets d'affichage.

Modification de la position

La manipulation la plus fondamentale de tout objet d'affichage consiste à le positionner à l'écran. Pour définir la position d'un objet d'affichage, changez ses propriétés `x` et `y`.

```
myShape.x = 17;
myShape.y = 212;
```

Le système de positionnement d'un objet d'affichage assimile la scène à un système de coordonnées cartésiennes (système de grille standard doté d'un axe horizontal x et d'un axe vertical y). L'origine du système de coordonnées (coordonnée 0,0 correspondant à l'intersection des axes x et y) figure dans le coin supérieur gauche de la scène. A partir de l'origine, les valeurs x sont positives vers la droite et négatives vers la gauche, tandis que, contrairement aux systèmes de graphes standard, les valeurs y sont positives vers le bas et négatives vers le haut. Par exemple, les lignes précédentes de code déplacent l'objet `myShape` vers la coordonnée x 17 (17 pixels sur la droite de l'origine) et la coordonnée y 212 (212 pixels sous l'origine).

Par défaut, lorsqu'un objet d'affichage est créé dans ActionScript, les propriétés `x` et `y` sont toutes deux définies sur 0, plaçant ainsi l'objet dans le coin supérieur gauche de son contenu parent.

Modification de la position relativement à la scène

Il est important de se rappeler que les propriétés `x` et `y` se réfèrent toujours à la position de l'objet d'affichage par rapport aux coordonnées 0,0 des axes de son objet d'affichage parent. Ainsi, pour une occurrence de Shape (par exemple un cercle) contenue dans une occurrence de Sprite, mettre à zéro les propriétés `x` et `y` de l'objet Shape revient à placer le cercle dans le coin supérieur gauche de l'objet Sprite, qui n'est pas forcément le coin supérieur gauche de la scène. Pour positionner un objet par rapport aux coordonnées globales de la scène, utilisez la méthode `globalToLocal()` de tout objet d'affichage afin de convertir les coordonnées globales (scène) en coordonnées locales (conteneur de l'objet d'affichage), comme suit :

```
// 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;
```

Vous pouvez également utiliser la méthode `localToGlobal()` de la classe DisplayObject pour convertir les coordonnées locales en coordonnées de la scène.

Création d'une interaction de type glisser-déposer

Un objet d'affichage est souvent déplacé pour créer une interaction de type glisser-déposer, de sorte que lorsque l'utilisateur clique sur un objet, celui-ci soit déplacé avec la souris jusqu'à ce que l'utilisateur relâche le bouton de la souris. Vous disposez de deux techniques pour créer une interaction de type glisser-déposer en ActionScript. Dans les deux cas, deux événements souris sont utilisés : lorsque l'utilisateur appuie sur le bouton de gauche (l'objet reçoit alors l'instruction de suivre le curseur de la souris) et lorsqu'il le relâche (l'objet reçoit alors l'instruction de cesser de suivre le curseur de la souris).

La première technique, basée sur la méthode `startDrag()`, est plus simple, mais plus limitée. Lorsque l'utilisateur appuie sur le bouton de la souris, la méthode `startDrag()` de l'objet d'affichage à faire glisser est appelée. Lorsque l'utilisateur relâche le bouton de la souris, la méthode `stopDrag()` est appelée.

```
// This code creates a drag-and-drop interaction using the startDrag()
// technique.
// square is a DisplayObject (e.g. 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);
```

Cette technique présente un désavantage relativement important : l'utilisation de `startDrag()` ne permet de faire glisser qu'un seul élément à la fois. Si un objet d'affichage est en cours de glissement et que la méthode `startDrag()` est appelée sur un autre objet d'affichage, le premier objet cesse immédiatement de suivre la souris. Par exemple, si la fonction `startDragging()` est modifiée comme indiqué, seul l'objet `circle` est déplacé par glissement, bien que la méthode `square.startDrag()` ait été appelée :

```
function startDragging(event:MouseEvent):void
{
    square.startDrag();
    circle.startDrag();
}
```

Puisqu'un seul objet à la fois peut être déplacé par glissement par le biais de `startDrag()`, la méthode `stopDrag()` peut être appelée sur n'importe quel objet d'affichage et arrête tout objet en cours de glissement.

Pour déplacer plusieurs objets d'affichage, ou pour éviter tout risque de conflit au cas où plusieurs objets seraient susceptibles d'utiliser `startDrag()`, il est préférable d'utiliser la technique de suivi de la souris pour créer l'effet de glisser-déposer. Avec cette technique, lorsque l'utilisateur appuie sur le bouton de la souris, une fonction est enregistrée en tant qu'écouteur de l'événement `mouseMove` de la scène. Cette fonction, qui est alors appelée à chaque déplacement de la souris, entraîne le saut de l'objet déplacé vers la coordonnée x, y de la souris. Une fois le bouton de la souris relâché, la fonction n'est plus enregistrée en tant qu'écouteur. Elle n'est donc plus appelée lorsque la souris est déplacée et l'objet cesse de suivre le curseur. Le code suivant illustre cette technique :

```
// This code creates a drag-and-drop interaction 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);
```

Outre le suivi du curseur par un objet d'affichage, l'interaction de type glisser-déposer entraîne le positionnement de l'objet déplacé à l'avant de l'affichage, créant ainsi une impression de flottement au-dessus de tous les autres objets. Supposons par exemple que deux objets, un cercle et un carré, soient soumis à une interaction de type glisser-déposer. Si le cercle figure sous le carré sur la liste d'affichage et que vous cliquez et faites glisser le cercle pour placer le curseur au-dessus du carré, il semble glisser derrière le carré, brisant ainsi l'illusion du glisser-déposer. Vous pouvez procéder de sorte que lorsque vous cliquez sur le cercle, il passe en première position dans la liste d'affichage et s'affiche ainsi par-dessus tout autre contenu.

Le code suivant (adapté de l'exemple précédent) crée une interaction de type glisser-déposer pour deux objets d'affichage, un cercle et un carré. Lorsque le bouton de la souris est pressé au-dessus de l'un d'eux, cet élément est amené en haut de la liste d'affichage de la scène, afin que l'élément déplacé passe toujours devant les autres. Tout nouveau code ou code modifié extrait du code précédent est imprimé en gras.

```
// 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);
```

Pour créer un effet plus sophistiqué, par exemple pour un jeu où des jetons ou des cartes passent d'une pile à l'autre, il est possible d'ajouter l'objet déplacé à la liste d'affichage de la scène lorsqu'il est « pris », puis de l'ajouter à une autre liste d'affichage (la « pile » sur laquelle il est déposé) lors du relâchement du bouton de souris.

Enfin, pour optimiser l'effet, vous pourriez appliquer un filtre Ombre portée à l'objet d'affichage lorsque vous cliquez dessus (en début de glissement) et supprimer l'ombre portée lorsque vous relâchez l'objet. Pour plus d'informations sur le filtre Ombre portée et tout autre filtre appliqué aux objets d'affichage en ActionScript, consultez le chapitre « Filtrage des objets d'affichage » à la page 363.

Défilement horizontal ou vertical des objets d'affichage

Si la taille d'un objet d'affichage est trop élevée pour la zone où vous souhaitez l'afficher, vous disposez de la propriété `scrollRect` pour définir la zone visible de l'objet d'affichage. Par ailleurs, en modifiant la propriété `scrollRect` en réponse à une action de l'utilisateur, vous pouvez entraîner un défilement vers la gauche ou la droite ou vers le haut ou le bas.

La propriété `scrollRect` est une occurrence de la classe Rectangle, qui combine les valeurs requises pour définir une zone rectangulaire en tant qu'objet unique. Pour définir initialement la zone visible de l'objet d'affichage, créez une occurrence de Rectangle et affectez-la à la propriété `scrollRect` de l'objet. Par la suite, pour obtenir un défilement horizontal ou vertical, il suffit de lire la propriété `scrollRect` dans une variable Rectangle séparée et de changer la propriété voulue (par exemple, modifier la propriété `x` de l'occurrence de Rectangle pour un défilement horizontal, ou sa propriété `y` pour un défilement vertical). Vous réaffectez ensuite cette occurrence de Rectangle à la propriété `scrollRect` pour avertir l'objet d'affichage du changement de valeur.

Par exemple, le code suivant définit la zone visible d'un objet TextField nommé `bigText` dont la hauteur est trop importante pour les dimensions du fichier SWF. Lorsque l'utilisateur clique sur les deux boutons nommés `up` et `down`, les fonctions appelées entraînent le défilement vertical du contenu de l'objet TextField en modifiant la propriété `y` de l'occurrence de Rectangle `scrollRect`.

```
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);
```

Comme le montre cet exemple, lorsque vous utilisez la propriété `scrollRect` d'un objet d'affichage, il est préférable de spécifier que Flash ou AIR doit mettre en cache le contenu de l'objet sous forme de bitmap, à l'aide de la propriété `cacheAsBitmap`. Ainsi, Flash Player et AIR n'ont pas à redessiner le contenu complet de l'objet d'affichage à chaque défilement de celui-ci, et peuvent utiliser le bitmap mis en cache pour afficher la portion concernée directement à l'écran. Pour plus d'informations, consultez la section « Mise en cache des objets d'affichage » à la page 308.

Manipulation de la taille et de l'échelle des objets

La taille d'un objet d'affichage peut être obtenue et modifiée de deux façons, en utilisant soit les propriétés de dimensions (`width` et `height`), soit les propriétés d'échelle (`scaleX` et `scaleY`).

Chaque objet d'affichage possède une propriété `width` et une propriété `height`, qui sont initialement définies sur la taille de l'objet en pixels. Vous pouvez lire les valeurs de ces propriétés pour mesurer la taille de l'objet d'affichage. Vous pouvez également stipuler de nouvelles valeurs pour modifier la taille de l'objet, comme suit :

```
// Resize a display object.
square.width = 420;
square.height = 420;

// Determine the radius of a circle display object.
var radius:Number = circle.width / 2;
```

Modifier les propriétés `height` ou `width` d'un objet d'affichage modifie l'échelle de ce dernier. En d'autres termes, son contenu est étiré ou comprimé en fonction de la taille de la nouvelle zone. Si l'objet d'affichage ne contient que des formes vectorielles, elles sont redessinées à la nouvelle échelle, sans perte de qualité. Tout élément graphique bitmap de l'objet d'affichage est mis à l'échelle au lieu d'être redessiné. Ainsi, une photo numérique dont la largeur et la hauteur augmentent de sorte à dépasser les dimensions réelles des informations relatives aux pixels de l'image est pixellisée, ce qui lui donne un aspect irrégulier.

Lorsque vous modifiez les propriétés `width` ou `height` d'un objet d'affichage, Flash Player et AIR mettent également à jour les propriétés `scaleX` ou `scaleY` de l'objet.

Remarque : *les objets TextField ne respectent pas ce comportement de mise à l'échelle. Les champs de texte doivent se redimensionner automatiquement pour gérer le retour à la ligne automatique et les tailles de police. Leurs valeurs scaleX et scaleY sont donc réinitialisées à 1 au terme du redimensionnement. Toutefois, si vous ajustez la valeur scaleX ou scaleY d'un objet TextField, les valeurs de largeur et de hauteur sont modifiées en fonction des valeurs de mise à l'échelle que vous indiquez.*

Ces propriétés représentent la taille relative de l'objet d'affichage par rapport à sa taille d'origine. Les propriétés `scaleX` et `scaleY` utilisent des valeurs exprimées sous forme de fractions (décimales) pour représenter le pourcentage. Par exemple, si la propriété `width` d'un objet d'affichage a été réduite à la moitié de sa largeur originale, la propriété `scaleX` de cet objet prendra la valeur `0,5`, soit 50 %. Si sa hauteur a doublé, sa propriété `scaleY` prendra la valeur `2`, soit 200 %.

```
// 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
```

Les changements de taille ne sont pas proportionnels. En d'autres termes, si vous modifiez la `hauteur` d'un carré, mais non sa `largeur`, ses proportions ne sont plus identiques et il devient un rectangle au lieu d'un carré. Si vous souhaitez modifier relativement la taille d'un objet d'affichage, vous pouvez définir les valeurs des propriétés `scaleX` et `scaleY` pour redimensionner l'objet, plutôt que définir les propriétés `width` ou `height`. Par exemple, ce code modifie la propriété `width` de l'objet d'affichage appelé `square`, puis modifie l'échelle verticale (`scaleY`) en fonction de l'échelle horizontale, afin que la taille du carré demeure proportionnelle.

```
// 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;
```

Contrôle de la distorsion lors de la mise à l'échelle

En règle générale, lorsqu'un objet d'affichage est mis à l'échelle (s'il est étiré horizontalement, par exemple), la distorsion résultante est répartie équitablement dans l'objet, de sorte que chaque partie soit soumise à un étirement identique. Pour les graphiques et éléments de conception, cette technique correspond probablement au résultat escompté. Toutefois, il est parfois préférable de garder le contrôle sur les parties de l'objet qui seront étirées et celles qui ne le seront pas. Un exemple courant est celui d'un bouton représenté par un rectangle aux angles arrondis. Lors d'une mise à l'échelle normale, les angles du bouton sont étirés, entraînant ainsi la modification de leur rayon lorsque le bouton est redimensionné.

Mais dans ce cas précis, il serait préférable de contrôler la mise à l'échelle, c'est-à-dire de pouvoir désigner certaines zones qui seront mises à l'échelle (les côtés) et celles qui ne le seront pas (les angles), afin que le changement d'échelle ne provoque pas de distorsion visible.

Vous disposez de la mise à l'échelle à 9 découpes (Echelle-9) pour créer des objets d'affichage dont vous contrôlez le mode de mise à l'échelle. La mise à l'échelle à 9 découpes permet de diviser l'objet d'affichage en neuf rectangles distincts (grille de 3 sur 3). Ces rectangles ne sont pas obligatoirement de la même taille, car l'utilisateur choisit l'emplacement des lignes de séparation. Tout contenu placé dans les quatre rectangles de coin (tels que les angles arrondis d'un bouton) n'est ni étiré, ni comprimé lors de la mise à l'échelle de l'objet d'affichage. Les rectangles placés en haut au centre et en bas au centre sont mis à l'échelle horizontalement, mais non verticalement, tandis que les rectangles centraux de gauche et de droite sont mis à l'échelle verticalement, mais non horizontalement. Le rectangle central est mis à l'échelle horizontalement et verticalement.

Ainsi, si vous créez un objet d'affichage et souhaitez qu'une partie du contenu ne subisse jamais de mise à l'échelle, il vous suffit de placer les lignes de séparation de la grille de mise à l'échelle à 9 découpes de telle sorte que ce contenu se trouve dans l'un des rectangles des angles.

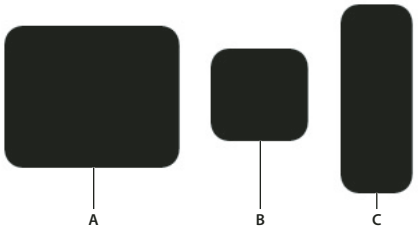
En ActionScript, il suffit de définir une valeur pour la propriété `scale9Grid` d'un objet d'affichage pour activer la mise à l'échelle à 9 découpes pour cet objet et définir la taille des rectangles de la grille. Vous utilisez une occurrence de la classe Rectangle en tant que valeur de la propriété `scale9Grid`, comme suit :

```
myButton.scale9Grid = new Rectangle(32, 27, 71, 64);
```

Les quatre paramètres du constructeur Rectangle sont la coordonnée x, la coordonnée y, la largeur et la hauteur. Dans cet exemple, l'angle supérieur gauche du rectangle est placé au point x : 32, y : 27 sur l'objet d'affichage appelé `myButton`. Le rectangle mesure 71 pixels de large et 64 pixels de haut (son bord droit correspond à la coordonnée x 103 sur l'objet d'affichage et son bord inférieur à la coordonnée y 92 sur l'objet d'affichage).

La zone figurant dans la région définie par l'occurrence de Rectangle représente le rectangle central de la grille Echelle-9. Les autres rectangles sont calculés par Flash Player et AIR en prolongeant les côtés de l'occurrence de Rectangle, comme indiqué :

Dans ce cas de figure, lorsque la mise à l'échelle du bouton augmente ou diminue, les angles arrondis ne sont ni étirés, ni comprimés, mais les autres zones sont ajustées en conséquence.



A. *myButton.width = 131;myButton.height = 106;* ***B.*** *myButton.width = 73;myButton.height = 69;* ***C.*** *myButton.width = 54;myButton.height = 141;*

Mise en cache des objets d'affichage

Que vous créiez une application ou des animations scriptées complexes, vous devez considérer la performance et l'optimisation, à mesure que la taille de vos conceptions Flash augmente. Lorsque votre contenu demeure statique (par exemple une occurrence de Shape rectangulaire), Flash et AIR ne l'optimisent pas. Par conséquent, lorsque vous modifiez la position du rectangle, Flash Player ou AIR redessine la totalité de l'occurrence de Shape.

Vous pouvez mettre en cache des objets d'affichage spécifiques pour améliorer les performances de votre fichier SWF. L'objet d'affichage est essentiellement une *surface*, c'est-à-dire une version bitmap des données vectorielles de l'occurrence, données non destinées à être considérablement modifiées tout au long de la vie de votre fichier SWF. Ainsi, les occurrences pour lesquelles la mise en cache est activée ne sont pas continuellement redessinées pendant la lecture du fichier SWF, et le rendu de ce dernier est plus rapide.

Remarque : *vous pouvez mettre à jour les données vectorielles au moment de la recréation de la surface. Ainsi, les données vectorielles mises en cache dans la surface ne doivent pas nécessairement être les mêmes pour l'ensemble du fichier SWF.*

Pour qu'un objet d'affichage mette en cache sa représentation sous forme de bitmap, il suffit d'activer sa propriété `cacheAsBitmap` (`true`). Flash Player ou AIR crée un objet de surface pour l'occurrence, correspondant à une image bitmap mise en cache et non à des données vectorielles. Si vous modifiez les limites de l'objet d'affichage, la surface est recréée et non redimensionnée. Les surfaces peuvent s'imbriquer dans d'autres surfaces. La surface copiera l'image bitmap sur sa surface parent. Pour plus d'informations, consultez la section « Activation de la mise en cache sous forme de bitmap » à la page 310.

Les propriétés `opaqueBackground` et `scrollRect` de la classe DisplayObject sont liées à la mise en cache sous forme de bitmap par le biais de la propriété `cacheAsBitmap`. Bien que ces trois propriétés soient indépendantes l'une de l'autre, `opaqueBackground` et `scrollRect` ne sont utiles que lorsqu'un objet est mis en cache sous forme de bitmap. Les avantages des propriétés `opaqueBackground` et `scrollRect` en termes de performances ne sont visibles que si `cacheAsBitmap` est définie sur `true`. Pour plus d'informations sur le défilement du contenu des objets d'affichage, consultez la section « Défilement horizontal ou vertical des objets d'affichage » à la page 304. Pour plus d'informations sur la définition d'un arrière-plan opaque, consultez la section « Définition d'un arrière-plan opaque » à la page 311.

Pour plus d'informations sur le masquage du canal alpha, qui demande que vous définissiez la propriété `cacheAsBitmap` sur `true`, consultez la section « Masquage des objets d'affichage » à la page 315.

Quand activer la mise en cache

L'activation de la mise en cache d'un objet d'affichage crée une surface dont les avantages sont multiples, par exemple pour accélérer le rendu des animations vectorielles complexes. Lorsque vous souhaitez activer la mise en cache, plusieurs scénarios sont disponibles. Il pourrait sembler avantageux de toujours activer la mise en cache pour améliorer les performances de votre fichier SWF. Cependant, dans certains cas, cette opération n'a aucun effet bénéfique et risque même parfois de ralentir le fichier. Cette section présente des cas où la mise en cache s'avère bénéfique, et d'autres où il est préférable d'utiliser les objets de façon normale.

Les performances générales des données mises en cache dépendent de la complexité des données vectorielles de vos occurrences, de la quantité de modifications et de la définition, ou non, de la propriété `opaqueBackground`. Si vous modifiez de petites zones, la différence entre l'utilisation d'une surface et celle de données vectorielles sera négligeable. Vous pouvez dans ce cas tester les deux scénarios avant de déployer votre application.

Quand utiliser la mise en cache sous forme de bitmap

Ce qui suit est une série de scénarios dans lesquels vous pouvez voir les bénéfices significatifs qui résultent de la mise en cache sous forme de bitmap.

- Image complexe d'arrière-plan : application qui contient une image d'arrière-plan complexe de données vectorielles (peut-être une image à laquelle vous avez appliqué la commande de traçage de bitmap ou illustration créée dans Adobe Illustrator®). Vous pouvez animer les caractères sur l'arrière-plan, ce qui ralentit l'animation parce que l'arrière-plan a besoin de continuellement régénérer les données vectorielles. Pour améliorer les performances, vous pouvez définir la propriété `opaqueBackground` de l'objet d'affichage d'arrière-plan sur `true`. L'arrière-plan est rendu en tant que bitmap et peut être redessiné rapidement pour que l'animation soit lue beaucoup plus vite.
- Champ de texte de défilement : application qui affiche une grande quantité de texte dans un champ de texte de défilement. Vous pouvez placer le champ de texte dans un objet d'affichage que vous définissez comme déroulant à l'aide de bornes de déroulement (propriété `scrollRect`). Ceci permet un déroulement de pixels rapide pour l'occurrence indiquée. Quand un utilisateur déroule l'occurrence d'objet d'affichage, Flash Player ou AIR fait défiler les pixels déroulés vers le haut et génère la zone nouvellement exposée au lieu de régénérer tout le champ de texte.
- Système de fenêtres : application comportant un système complexe de chevauchement de fenêtres. Chaque fenêtre peut être ouverte ou fermée (par exemple, les fenêtres du navigateur Web). Si vous marquez chaque fenêtre en tant que surface (en définissant la propriété `cacheAsBitmap` sur `true`), chaque fenêtre est isolée et mise en cache. Les utilisateurs peuvent faire glisser les fenêtres de manière à ce qu'elles se chevauchent. Chaque fenêtre n'a pas besoin de régénérer le contenu vectoriel.
- Masquage du canal alpha : si vous utilisez le masquage du canal alpha, vous devez définir la propriété `cacheAsBitmap` sur `true`. Pour plus d'informations, consultez la section « Masquage des objets d'affichage » à la page 315.

Activer la mise en cache sous forme de bitmap dans tous ces scénarios améliore la réactivité et l'interactivité de l'application en optimisant les graphiques vectoriels.

Par ailleurs, lorsque vous appliquez un filtre à un objet d'affichage, `cacheAsBitmap` est automatiquement définie sur `true`, même si vous l'avez explicitement définie sur `false`. Si vous supprimez tous les filtres appliqués à l'objet d'affichage, la propriété `cacheAsBitmap` retrouve la valeur précédemment définie.

Quand éviter d'utiliser la mise en cache sous forme de bitmap

Un mauvais usage de cette fonctionnalité peut avoir un effet négatif sur votre fichier SWF. Avant d'utiliser la mise en cache sous forme de bitmap, tenez compte des considérations suivantes :

- N'abusez pas des surfaces (objets d'affichage avec mise en cache activée). Chaque surface utilise plus de mémoire qu'un objet d'affichage standard, ce qui signifie que vous ne devez activer les surfaces que lorsqu'il est nécessaire d'améliorer les performances de rendu.

  Un bitmap caché utilise beaucoup plus de mémoire qu'un objet d'affichage standard. Par exemple, si la taille d'une occurrence de Sprite sur la scène correspond à 250 pixels sur 250 pixels, elle est susceptible d'utiliser en cache 250 Ko au lieu d'1 Ko pour une occurrence de Sprite standard (non en cache).
- Evitez de zoomer dans les surfaces cachées. Si vous abusez de la mise en cache sous forme de bitmap, une grande quantité de mémoire sera occupée (voir la puce précédente), surtout si vous zoomez sur le contenu.
- Utilisez des surfaces essentiellement non statiques (sans animation) pour les occurrences d'objets d'affichage. Vous pouvez faire glisser ou déplacer l'occurrence, mais son contenu ne doit pas être animé ni subir de nombreuses modifications. (Les animations ou les contenus qui changent sont plus fréquents lorsqu'une occurrence de MovieClip contient une animation ou une occurrence de Video.) Par exemple, si vous faites pivoter ou transformez une occurrence, la différence entre la surface et les données vectorielles rend le traitement difficile et affecte le fichier SWF.
- Panacher des surfaces avec des données vectorielles accroît la charge de traitement de Flash Player et AIR (et quelquefois de l'ordinateur). Dans la mesure du possible, regroupez les surfaces, en particulier lorsque vous créez des applications à fenêtres.

Activation de la mise en cache sous forme de bitmap

Pour activer la mise en cache d'un objet d'affichage sous forme de bitmap, définissez sa propriété `cacheAsBitmap` sur `true` :

```
mySprite.cacheAsBitmap = true;
```

Après avoir défini la propriété `cacheAsBitmap` sur `true`, vous remarquerez que l'objet d'affichage accroche automatiquement les pixels sur les coordonnées entières. Lorsque vous testez le fichier SWF, vous devriez remarquer que le rendu d'une animation associée à une image vectorielle complexe est bien plus rapide.

Une surface (bitmap en cache) n'est pas créée même quand `cacheAsBitmap` est défini sur `true` s'il se produit l'un ou l'autre des événements suivants :

- Le bitmap fait plus de 2 880 pixels en hauteur ou en largeur.
- Il est impossible d'allouer de la mémoire pour l'image bitmap.

Définition d'un arrière-plan opaque

Vous pouvez définir un arrière-plan opaque pour un objet d'affichage. Par exemple, si l'arrière-plan de votre fichier SWF contient une illustration vectorielle complexe, vous pouvez définir la propriété `opaqueBackground` sur une couleur donnée (en général la même couleur que la scène). La couleur est stipulée sous forme de nombre (généralement une valeur hexadécimale). L'arrière-plan est alors considéré comme un bitmap, ce qui permet d'optimiser les performances.

Lorsque vous définissez `cacheAsBitmap` sur `true` et la propriété `opaqueBackground` sur une couleur donnée, la propriété `opaqueBackground` assure l'opacité du bitmap interne et un rendu plus rapide. Si vous ne définissez pas `cacheAsBitmap` sur `true`, la propriété `opaqueBackground` ajoute une forme carrée vectorielle opaque à l'arrière-plan de l'objet d'affichage. Elle ne crée pas automatiquement un bitmap.

L'exemple suivant décrit comment définir l'arrière-plan d'un objet d'affichage pour optimiser les performances :

```
myShape.cacheAsBitmap = true;
myShape.opaqueBackground = 0xFF0000;
```

Dans ce cas, la couleur d'arrière-plan de l'objet Shape appelé `myShape` est définie sur rouge (`0xFF0000`). Si l'on suppose que l'occurrence de Shape contient un triangle vert, sur une scène dotée d'un fond blanc le résultat sera un triangle vert sur le fond rouge défini par le cadre de sélection de l'occurrence de Shape (le rectangle qui entoure entièrement Shape).

Ce code serait bien entendu plus logique s'il était utilisé avec une scène dotée d'un arrière-plan rouge uni. Si l'arrière-plan était d'une autre couleur, celle-ci remplacerait le rouge. Par exemple, dans un fichier SWF doté d'un arrière-plan blanc, la propriété `opaqueBackground` serait probablement définie sur `0xFFFFFF`, soit un blanc pur.

Application de modes de fondu

Les modes de fondu supposent d'associer les couleurs d'une image (l'image de base) à celles d'une autre image (l'image mélangée) afin de produire une troisième image, qui est celle qui sera en fait affichée à l'écran. Chaque valeur de pixels d'une image est traitée avec la valeur de pixels correspondante de l'autre image afin d'obtenir un résultat présentant une valeur de pixels de position identique.

Tous les objets d'affichage possèdent une propriété `blendMode` qui peut être définie sur l'un des modes de fondu ci-dessous. Les valeurs ci-dessous sont des constantes définies dans la classe BlendMode. Vous pouvez également utiliser les valeurs String (entre parenthèses) correspondant aux valeurs réelles des constantes.

- `BlendMode.ADD` (`"add"`) : s'utilise couramment pour créer un effet de dissolution animée entre deux images en éclaircissant progressivement leurs couleurs.
- `BlendMode.ALPHA` (`"alpha"`) : s'utilise couramment pour appliquer la transparence de l'avant-plan à l'arrière-plan.
- `BlendMode.DARKEN` (`"darken"`) :s'utilise couramment pour superposer un type.
- `BlendMode.DIFFERENCE` (`"difference"`) : s'utilise couramment pour créer des couleurs plus vives.
- `BlendMode.ERASE` (`"erase"`) : s'utilise couramment pour découper (effacer) une partie de l'arrière-plan à l'aide de l'alpha d'avant-plan.
- `BlendMode.HARDLIGHT` (`"hardlight"`) : s'utilise couramment pour créer des effets d'ombrage.
- `BlendMode.INVERT` (`"invert"`) : permet d'inverser l'arrière-plan.

- `BlendMode.LAYER` (`"layer"`) : permet d'imposer la création d'un tampon provisoire en vue de la précomposition d'un objet d'affichage spécifique.
- `BlendMode.LIGHTEN` (`"lighten"`) : s'utilise couramment pour superposer un type.
- `BlendMode.MULTIPLY` (`"multiply"`) : s'utilise couramment pour créer des ombres et des effets de profondeur.
- `BlendMode.NORMAL` (`"normal"`) : permet aux valeurs des pixels de l'image mélangée de supplanter celles de l'image de base.
- `BlendMode.OVERLAY` (`"overlay"`) : s'utilise couramment pour créer des effets d'ombrage.
- `BlendMode.SCREEN` (`"screen"`) : s'utilise couramment pour créer des mises en surbrillance et des halos.
- `BlendMode.SHADER` (`"shader"`) : permet d'indiquer l'utilisation d'un shader de Pixel Bender pour créer un effet de mélange personnalisé. Pour plus d'informations sur l'utilisation des shaders, consultez le chapitre « Utilisation des shaders de Pixel Bender » à la page 395.
- `BlendMode.SUBTRACT` (`"subtract"`) : s'utilise couramment pour créer un effet de dissolution animée entre deux images en assombrissant progressivement leurs couleurs.

Réglage des couleurs DisplayObject

Vous pouvez utiliser les méthodes de la classe ColorTransform (flash.geom.ColorTransform) pour régler la couleur d'un objet d'affichage. Chaque objet d'affichage possède une propriété `transform`, qui est une occurrence de la classe Transform et contient des informations relatives aux diverses transformations appliquées à l'objet d'affichage (rotation, changements d'échelle ou de position, etc.). Outre ces informations sur les transformations géométriques, la classe Transform comprend également une propriété `colorTransform`, qui est une occurrence de la classe ColorTransform et permet de régler les couleurs de l'objet d'affichage Pour accéder aux informations de transformation d'un objet d'affichage, utilisez le code suivant :

```
var colorInfo:ColorTransform = myDisplayObject.transform.colorTransform;
```

Après avoir créé une occurrence de ColorTransform, vous pouvez lire les valeurs de ses propriétés pour connaître les transformations de couleur qui lui ont déjà été appliquées, et vous pouvez modifier ces valeurs pour changer les couleurs de l'objet. Pour mettre à jour l'objet d'affichage après toute modification, vous devez réaffecter l'occurrence de ColorTransform à la propriété `transform.colorTransform`.

```
var colorInfo:ColorTransform = my DisplayObject.transform.colorTransform;

// Make some color transformations here.

// Commit the change.
myDisplayObject.transform.colorTransform = colorInfo;
```

Définition des valeurs de couleur à l'aide du code

La propriété `color` de la classe ColorTransform permet d'affecter une valeur de couleur rouge, vert, bleu (RVB) déterminée à l'objet d'affichage. L'exemple suivant utilise la propriété `color` pour changer en bleu la couleur de l'objet d'affichage `square` lorsque l'utilisateur clique sur le bouton `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);
```

Notez que si vous changez la couleur d'un objet d'affichage à l'aide de la propriété `color`, c'est la couleur de l'objet entier qui est modifiée, même s'il comportait plusieurs couleurs à l'origine. Par exemple, si un objet d'affichage contient un cercle vert en arrière-plan d'un texte noir, le choix du rouge pour la propriété `color` de l'occurrence de ColorTransform associée à cet objet transformera intégralement l'objet en rouge, cercle et texte compris (le texte ne sera donc plus lisible sur le fond de la même couleur).

Modification des effets de couleur et de luminosité à l'aide du code

Supposons qu'un objet d'affichage comporte plusieurs couleurs (par exemple une photo numérique) et que vous n'ayez pas l'intention de modifier la couleur de l'ensemble de l'objet, mais uniquement celle d'un de ses éléments sur la base des couleurs existantes. Dans ce scénario, la classe ColorTransform comprend une série de propriétés de multiplication et de dominante permettant d'effectuer ce type de réglage. Les propriétés de multiplication, appelées `redMultiplier`, `greenMultiplier`, `blueMultiplier` et `alphaMultiplier`, fonctionnent comme des filtres photographiques de couleur (ou des lunettes à verres de couleur) et amplifient ou diminuent certaines couleurs de l'objet d'affichage. Les propriétés de dominante (`redOffset`, `greenOffset`, `blueOffset` et `alphaOffset`) permettent d'ajouter à l'objet une quantité supplémentaire d'une certaine couleur, ou d'indiquer la valeur minimale que peut avoir une couleur particulière.

Ces propriétés de multiplication et de dominante sont identiques aux réglages de couleurs avancés relatifs aux symboles de clips dans l'outil de programmation Flash lorsque vous sélectionnez Paramètres avancés dans le menu déroulant Couleur de l'Inspecteur des propriétés.

Le code suivant charge une image JPEG et lui applique une transformation de couleur, qui ajuste les canaux rouge et vert lorsque le pointeur de la classe se déplace le long des axes x et y. Dans ce cas précis, comme aucune valeur de dominante n'est spécifiée, les valeurs de chaque canal colorimétrique seront un pourcentage de la valeur de couleur originale de l'image, c'est-à-dire que la valeur maximale de rouge ou de vert d'un pixel donné sera la quantité originale de vert ou de rouge de ce pixel.

```
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);
```

Rotation des objets

La propriété `rotation` permet de faire pivoter les objets d'affichage. Vous pouvez lire cette valeur pour savoir si un objet a été soumis à une rotation. Vous pouvez également la définir sur un nombre (exprimé en degrés) représentant le montant de rotation à appliquer à l'objet. Par exemple, cette ligne de code fait pivoter l'objet `square` de 45 degrés (un huitième de tour complet) :

```
square.rotation = 45;
```

Vous pouvez également faire pivoter un objet d'affichage par le biais d'une matrice de transformation, décrite dans « Utilisation de la géométrie » à la page 350.

Application d'effets de fondu à des objets

Vous pouvez contrôler la transparence d'un objet d'affichage de sorte à le rendre partiellement (ou totalement) transparent, ou modifier la transparence pour créer une impression d'apparition ou de disparition en fondu de l'objet. La propriété `alpha` de la classe DisplayObject définit la transparence (ou plus précisément l'opacité) d'un objet d'affichage. La propriété `alpha` peut être définie sur n'importe quelle valeur comprise entre 0 et 1, sachant que 0 correspond à une transparence totale et 1 à une opacité totale. Par exemple, le code suivant rend l'objet `myBall` transparent à 50 % lors d'un clic de souris :

```
function fadeBall(event:MouseEvent):void
{
    myBall.alpha = .5;
}
myBall.addEventListener(MouseEvent.CLICK, fadeBall);
```

Vous pouvez également modifier la transparence d'un objet d'affichage en utilisant les réglages de couleur proposés par la classe ColorTransform. Pour plus d'informations, consultez la section « Réglage des couleurs DisplayObject » à la page 312.

Masquage des objets d'affichage

Vous pouvez utiliser un objet d'affichage comme masque pour créer un « trou » laissant apparaître le contenu d'un autre objet.

Définition d'un masque

Pour indiquer qu'un objet d'affichage sera le masque d'un autre objet d'affichage, définissez l'objet masque comme propriété `mask` de l'objet à masquer :

```
// Make the object maskSprite be a mask for the object mySprite.
mySprite.mask = maskSprite;
```

L'objet d'affichage masqué est révélé sous toutes les zones opaques (non transparentes) de l'objet d'affichage servant de masque. Par exemple, le code suivant crée une occurrence de Shape qui contient un carré rouge de 100 pixels sur 100 et une occurrence de Sprite contenant un cercle bleu d'un rayon de 25 pixels. Lorsque l'utilisateur clique sur le cercle, il devient le masque du carré, de sorte que l'unique partie du carré affichée est la section couverte par la partie pleine du cercle. En d'autres termes, seul un cercle rouge est visible.

```
// 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);
```

L'objet d'affichage qui sert de masque peut être glissé, animé, redimensionné dynamiquement et peut utiliser plusieurs formes au sein d'un masque unique. Il n'est pas nécessaire que l'objet qui fait office de masque soit ajouté à la liste d'affichage. Toutefois, pour que l'objet servant de masque soit mis à l'échelle lors de la mise à l'échelle de la scène ou pour activer l'interaction utilisateur avec le masque (opération de type glisser et redimensionner contrôlée par l'utilisateur, par exemple), vous devez l'ajouter à la liste d'affichage. L'indice de profondeur (z-index, pour l'ordre de superposition) des objets d'affichage n'a pas d'importance, dès lors que l'objet masque est ajouté à la liste d'affichage. (L'objet servant de masque ne s'affiche pas à l'écran, sauf en tant que masque.) Si l'objet servant de masque est une occurrence de MovieClip dotée de plusieurs images, il lit toutes les images de son scénario, comme il le ferait s'il ne faisait pas office de masque. Pour supprimer un masque, définissez la propriété `mask` sur `null` :

```
// remove the mask from mySprite
mySprite.mask = null;
```

Il est impossible d'utiliser un masque pour en masquer un autre. Il est impossible de définir la propriété `_alpha` d'un objet d'affichage servant de masque. Seuls les remplissages sont utilisés dans un objet d'affichage employé comme masque ; les traits sont ignorés.

A propos du masquage des polices de périphérique

Vous pouvez utiliser un objet d'affichage pour masquer le texte défini dans une police de périphérique. Dans ce cas, le cadre de délimitation rectangulaire du masque est utilisé comme forme de masquage. Ainsi, si vous créez un masque objet d'affichage non rectangulaire pour un texte de police de périphérique, le masque qui apparaît dans le fichier SWF prend la forme du cadre de délimitation rectangulaire du masque, et non celle du masque en tant que tel.

Masquage du canal alpha

Le masquage du canal alpha est pris en charge si le masque et les objets d'affichage masqués utilisent la mise en cache sous forme de bitmap, comme illustré ci-après :

```
// maskShape is a Shape instance which includes a gradient fill.
mySprite.cacheAsBitmap = true;
maskShape.cacheAsBitmap = true;
mySprite.mask = maskShape;
```

Une application du masquage du canal alpha consiste par exemple à appliquer un filtre à l'objet masque indépendamment d'un filtre appliqué à l'objet d'affichage masqué.

Dans l'exemple suivant, un fichier d'image externe est chargé sur la scène. Cette image (ou, plus précisément, l'occurrence de Loader dans laquelle elle est chargée) correspondra à l'objet d'affichage masqué. Un ovale dégradé (centre noir uni dont les bords deviennent progressivement transparents) est dessiné sur l'image. Il s'agit-là du masque alpha. La mise en cache sous forme de bitmap est activée pour les deux objets d'affichage. L'ovale est défini en tant que masque de l'image, puis peut être déplacé.

```
// 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);
```

Animation des objets

L'animation consiste à faire bouger un élément ou à le faire progressivement évoluer. Les animations par script représentent un élément fondamental des jeux vidéo, et elles sont aussi fréquemment utilisées pour obtenir un résultat plus séduisant et ajouter des interactions à d'autres applications.

Une animation par script est régie par un principe de base : il doit se produire une évolution et cette dernière doit être divisée en incréments, au fil du temps. Il est facile de répéter une opération en ActionScript, à l'aide d'une simple boucle. Toutefois, une boucle exécute toutes ses itérations avant de mettre à jour l'affichage. Pour créer une animation par script, vous devez écrire un code ActionScript qui exécute plusieurs fois une action au fil du temps et met également à jour l'écran à chaque exécution.

Supposons par exemple que vous souhaitez créer une animation simple, telle qu'une balle qui traverse l'écran. ActionScript comprend un mécanisme simple qui permet de suivre le passage du temps et de mettre à jour l'écran en conséquence. En d'autres termes, vous pourriez écrire un code qui déplace la balle d'un petit incrément à chaque fois jusqu'à ce qu'elle atteigne sa destination. Après chaque déplacement, l'écran serait mis à jour, afin que l'utilisateur puisse visualiser le mouvement à l'écran.

D'un point de vue pratique, il est logique de synchroniser l'animation par script avec la cadence du fichier SWF (en d'autres termes, de modifier une animation à chaque fois qu'une nouvelle image s'affiche ou devrait s'afficher), puisque cela permet de définir la fréquence des mises à jour de l'écran par Flash Player ou AIR. Chaque objet d'affichage possède un événement `enterFrame` qui est diffusé en fonction de la cadence d'affichage du fichier SWF (un événement par image). La plupart des développeurs qui créent une animation par script utilisent l'événement `enterFrame` pour générer des actions répétées au fil du temps. Vous pourriez écrire du code qui écoute l'événement `enterFrame`, déplace la balle animée d'un incrément déterminé à chaque image et, lorsque l'écran est mis à jour (à chaque image), la balle est redessinée à sa nouvelle position, créant ainsi un mouvement.

Remarque : *une autre technique pour exécuter une action de manière répétitive dans le temps consiste à utiliser la classe Timer. Une occurrence de Timer déclenche une notification d'événement après un délai horaire donné. Il est donc possible d'écrire du code effectuant une animation sur la base de l'événement timer de la classe Timer, en définissant un intervalle très court (une fraction de seconde). Pour plus d'informations sur l'utilisation de la classe Timer, consultez la section* « *Contrôle des intervalles temporels* » *à la page 138.*

Dans l'exemple suivant, une occurrence circulaire de Sprite, appelée `circle`, est créée sur la scène. Lorsque l'utilisateur clique sur le cercle, une séquence animée par script débute et entraîne un fondu de `circle`(sa propriété `alpha` diminue) jusqu'à ce qu'il soit complètement transparent :

```
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);
```

Lorsque l'utilisateur clique sur le cercle, la fonction `fadeCircle()` est enregistrée en tant qu'écouteur de l'événement `enterFrame`. En d'autres termes, elle commence à être appelée une fois par image. Cette fonction provoque un fondu de l'objet `circle` en changeant sa propriété `alpha`, si bien qu'à chaque nouvelle image la valeur de la propriété `alpha` du cercle décroît de 0,05 (soit 5 %) et l'écran est actualisé. Au fil du temps, lorsque la valeur `alpha` correspond à 0 (auquel cas `circle` est complètement transparent), la fonction `fadeCircle()` est supprimée des écouteurs d'événement et l'animation s'arrête.

Le même code permet, par exemple, de créer un mouvement animé au lieu d'un fondu. En substituant une autre propriété à `alpha` dans la fonction qui écoute l'événement `enterFrame`, cette propriété est animée. Par exemple, remplacer la ligne

```
circle.alpha -= .05;
```

par la ligne

```
circle.x += 5;
```

anime la propriété `x`. Le cercle se déplace alors vers la droite de la scène. La condition qui arrête l'animation peut être modifiée de sorte à arrêter l'animation (en d'autres termes, annuler l'enregistrement de l'écouteur `enterFrame`) lorsque la coordonnée x appropriée est atteinte.

Chargement dynamique du contenu d'affichage

Les éléments d'affichage externes suivants peuvent être chargés dans une application ActionScript 3.0 :

- Fichier SWF programmé dans ActionScript 3.0 : ce fichier peut correspondre à Sprite, MovieClip ou toute classe qui étend Sprite.
- Fichier d'image : tels que les fichiers JPG, PNG et GIF.
- Fichier AVM1 SWF : fichier SWF écrit en ActionScript 1.0 ou 2.0.

Vous chargez ces ressources par le biais de la classe Loader.

Chargement d'objets d'affichage

Les objets Loader permettent de charger des fichiers SWF et des fichiers graphiques dans une application. La classe Loader est une sous-classe de la classe DisplayObjectContainer. La liste d'affichage d'un objet Loader ne comporte qu'un seul objet d'affichage enfant : l'objet d'affichage qui représente le fichier SWF ou graphique qu'il charge. Lorsque vous ajoutez un objet Loader à la liste d'affichage, comme dans le code ci-dessous, vous ajoutez également l'objet d'affichage enfant chargé à la liste d'affichage, une fois le chargement effectué :

```
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
this.addChild(pictLdr);
```

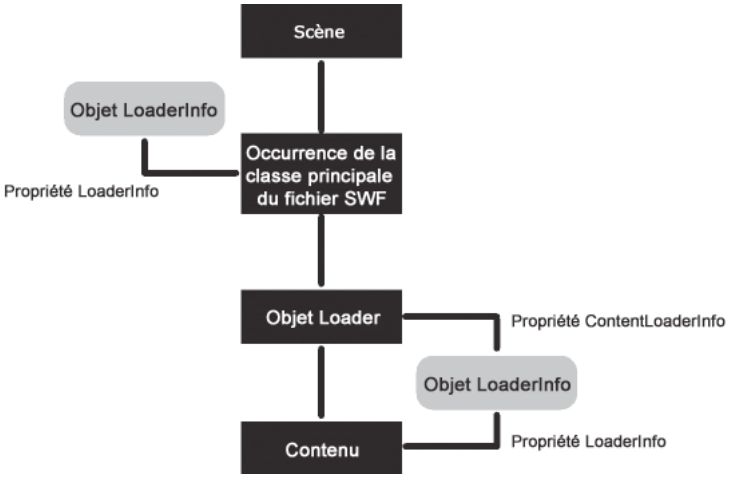
Lorsque le fichier SWF ou l'image sont chargés, vous pouvez transférer l'objet d'affichage chargé dans un autre conteneur d'objets d'affichage, tel que l'objet `container` de la classe DisplayObjectContainer dans l'exemple illustré :

```
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);
}
```

Surveillance de la progression du chargement

Lorsque le chargement du fichier débute, un objet LoaderInfo est créé. Un objet LoaderInfo fournit diverses informations sur le chargement : progression, adresses URL du chargeur et du chargé, nombre d'octets total de l'objet multimédia et dimensions nominales (hauteur et largeur) de celui-ci. Par ailleurs, un objet LoaderInfo distribue les événements qui permettent de suivre la progression du chargement.

Le diagramme suivant présente les diverses utilisations de l'objet LoaderInfo, pour l'occurrence de la classe principale du fichier SWF, pour un objet Loader et pour un objet chargé par ce dernier :



Vous pouvez accéder à l'objet LoaderInfo en tant que propriété de l'objet Loader et de l'objet d'affichage chargé. Dès que le chargement débute, vous pouvez accéder à l'objet LoaderInfo par le biais de la propriété `contentLoaderInfo` de l'objet Loader. Lorsque le chargement de l'objet d'affichage est terminé, vous pouvez également accéder à l'objet LoaderInfo en tant que propriété de cet objet chargé par le biais de la propriété `loaderInfo` de l'objet d'affichage. La propriété `loaderInfo` de l'objet d'affichage chargé se réfère au même objet LoaderInfo que la propriété `contentLoaderInfo` de l'objet Loader. En d'autres termes, un objet LoaderInfo est partagé entre un objet chargé et l'objet Loader qui l'a chargé (entre le chargeur et le chargé).

Pour accéder aux propriétés du contenu chargé, il est recommandé d'ajouter un écouteur d'événement à l'objet LoaderInfo, comme indiqué dans le code suivant :

```
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;
}
```

Pour plus d'informations, consultez le chapitre « Gestion des événements » à la page 254.

Définition du contexte de chargement

Lorsque vous chargez un fichier externe dans Flash Player ou AIR par le biais de la méthode `load()` ou `loadBytes()` de la classe Loader, vous pouvez indiquer le paramètre `context`. Ce paramètre est un objet LoaderContext.

La classe LoaderContext comporte trois propriétés qui permettent de définir le contexte d'utilisation du contenu chargé :

- `checkPolicyFile` : utilisez cette propriété uniquement pour le chargement d'un fichier image (pas pour un fichier SWF). Si vous définissez cette propriété sur `true`, l'objet Loader vérifie si le serveur d'origine héberge un fichier de régulation (voir « Contrôles de site Web (fichiers de régulation) » à la page 721). Cette opération n'est requise que si le contenu émane de domaines autres que celui du fichier SWF qui contient l'objet Loader. Si le serveur accorde une autorisation au domaine de Loader, le code ActionScript extrait des fichiers SWF du domaine de Loader peut accéder aux données de l'image chargée. En d'autres termes, vous pouvez utiliser la commande `BitmapData.draw()` pour accéder aux données de l'image chargées.

  Notez qu'un fichier SWF extrait d'un autre domaine que celui de l'objet Loader peut appeler `Security.allowDomain()` pour autoriser un domaine déterminé.

- `securityDomain` : utilisez cette propriété uniquement pour le chargement d'un fichier SWF (pas pour une image). Cette propriété peut être appelée pour un fichier SWF provenant d'un autre domaine que celui du fichier qui contient l'objet Loader. Lorsque vous indiquez cette option, Flash Player vérifie l'existence d'un fichier de régulation et, s'il existe, les fichiers SWF des domaines autorisés dans ce fichier peuvent utiliser des opérations de programmation croisée avec le contenu du fichier SWF chargé. Vous pouvez stipuler `flash.system.SecurityDomain.currentDomain` en tant que paramètre.

- `applicationDomain` : utilisez cette propriété uniquement lors du chargement d'un fichier SWF écrit dans ActionScript 3.0 (et non une image ou un fichier SWF écrit dans ActionScript 1.0 ou 2.0). Lorsque vous chargez un fichier, vous devez indiquer que le fichier doit être inclus dans le même domaine d'application que l'objet Loader en attribuant au paramètre `applicationDomain` la valeur `flash.system.ApplicationDomain.currentDomain`. Si vous placez le fichier SWF chargé dans le même domaine d'application, vous pourrez accéder directement à ses classes, ce qui s'avère utile si vous chargez un fichier SWF contenant des média intégrés, auxquels vous pouvez accéder via les noms de classes associés. Pour plus d'informations, consultez la section « Utilisation de la classe ApplicationDomain » à la page 666.

Exemple de vérification d'un fichier de régulation lors du chargement d'une image bitmap provenant d'un autre domaine :

```
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);
```

Exemple de vérification d'un fichier de régulation lors du chargement d'un fichier SWF à partir d'un autre domaine, dans le but de placer ce fichier dans la même Sandbox de sécurité que l'objet Loader. Par ailleurs, le code ajoute les classes du fichier SWF chargé dans le même domaine d'application que celui de l'objet Loader :

```
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);
```

Pour plus d'informations, consultez la classe LoaderContext dans le Guide de référence du langage et des composants ActionScript 3.0.

Exemple : SpriteArranger

L'exemple SpriteArranger est basé sur l'exemple Geometric Shapes décrit dans un autre chapitre (voir « Exemple : GeometricShapes » à la page 126).

L'exemple SpriteArranger illustre divers concepts de gestion des objets d'affichage :

- Extension des classes d'objet d'affichage
- Ajout d'objets à la liste d'affichage
- Superposition des objets d'affichage et utilisation des conteneurs d'objets d'affichage
- Réponse aux événements d'objet d'affichage
- Utilisation des propriétés et méthodes des objets d'affichage

Pour obtenir les fichiers d'application associés à cet exemple, voir www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application SpriteArranger résident dans le dossier Examples/SpriteArranger. L'application se compose des fichiers suivants :

Fichier	Description
SpriteArranger.mxml ou SpriteArranger.fla	Fichier d'application principal dans Flash (FLA) ou Flex (MXML).
com/example/programmingas3/SpriteArranger/CircleSprite.as	Classe définissant un type d'objet Sprite qui assure le rendu d'un cercle à l'écran.
com/example/programmingas3/SpriteArranger/DrawingCanvas.as	Classe définissant le rectangle de la zone de dessin, soit un conteneur d'objets d'affichage comportant des objets GeometricSprite.
com/example/programmingas3/SpriteArranger/SquareSprite.as	Classe définissant un type d'objet Sprite qui assure le rendu d'un carré à l'écran.
com/example/programmingas3/SpriteArranger/TriangleSprite.as	Classe définissant un type d'objet Sprite qui assure le rendu d'un triangle à l'écran.

Fichier	Description
com/example/programmingas3/SpriteArranger/GeometricSprite.as	Classe qui étend l'objet Sprite, utilisé pour définir une forme à l'écran. CircleSprite, SquareSprite et TriangleSprite étendent chacun cette classe.
com/example/programmingas3/geometricshapes/IGeometricShape.as	Interface de base qui définit les méthodes à implémenter par toutes les classes de formes géométriques.
com/example/programmingas3/geometricshapes/IPolygon.as	Interface qui définit les méthodes à implémenter par les classes de forme géométrique dotées de plusieurs côtés.
com/example/programmingas3/geometricshapes/RegularPolygon.as	Type de forme géométrique dont les côtés sont de longueur égale et positionnés symétriquement autour du centre de la forme.
com/example/programmingas3/geometricshapes/Circle.as	Type de forme géométrique qui définit un cercle.
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	Sous-classe de RegularPolygon qui définit un triangle dont tous les côtés ont la même longueur.
com/example/programmingas3/geometricshapes/Square.as	Sous-classe de RegularPolygon qui définit un rectangle dont les quatre côtés ont la même longueur.
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	Classe qui contient une « méthode usine » permettant de créer des formes d'une taille et d'un type de forme donnés.

Définition des classes SpriteArranger

L'application SpriteArranger permet à l'utilisateur d'ajouter divers objets d'affichage au rectangle de la zone de dessin à l'écran.

La classe DrawingCanvas définit une zone de dessin, soit un type de conteneur d'objets d'affichage, à laquelle l'utilisateur peut ajouter des formes à l'écran. Ces formes sont des occurrences de l'une des sous-classes de la classe GeometricSprite.

Classe DrawingCanvas

La classe DrawingCanvas étend la classe Sprite et cet héritage est défini dans la déclaration de classe DrawingCanvas, comme suit :

```
public class DrawingCanvas extends Sprite
```

La classe Sprite est une sous-classe des classes DisplayObjectContainer et DisplayObject et la classe DrawingCanvas utilise les méthodes et propriétés de ces dernières.

La méthode constructeur `DrawingCanvas()` définit un objet Rectangle, `bounds`, qui est une propriété utilisée ultérieurement lors du tracé du contour de la zone de dessin. Elle appelle ensuite la méthode `initCanvas()`, comme suit :

```
this.bounds = new Rectangle(0, 0, w, h);
initCanvas(fillColor, lineColor);
```

Comme l'indique l'exemple suivant, la méthode `initCanvas()` définit diverses propriétés de l'objet DrawingCanvas, transmises en tant qu'arguments à la méthode constructeur :

```
this.lineColor = lineColor;
this.fillColor = fillColor;
this.width = 500;
this.height = 200;
```

La méthode `initCanvas()` appelle ensuite la méthode `drawBounds()` qui trace le rectangle de la zone de dessin à l'aide de la propriété `graphics` de la classe DrawingCanvas. La propriété `graphics` est héritée de la classe 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();
```

Les autres méthodes de la classe DrawingCanvas, indiquées ci-après, sont appelées en fonction des interactions de l'utilisateur avec l'application :

- Les méthodes `addShape()` et `describeChildren()`, décrites à la section « Ajout d'objets d'affichage au rectangle de la zone de dessin » à la page 325
- Les méthodes `moveToBack()`, `moveDown()`, `moveToFront()` et `moveUp()`, décrites à la section « Réorganisation de l'ordre de superposition des objets d'affichage » à la page 327
- La méthode `onMouseUp()`, décrite à la section « Cliquer-déplacer un objet d'affichage » à la page 326

Classe GeometricSprite et ses sous-classes

Chaque objet d'affichage susceptible d'être ajouté par l'utilisateur au rectangle de la zone de dessin est une occurrence de l'une des sous-classes suivantes de la classe GeometricSprite :

- CircleSprite
- SquareSprite
- TriangleSprite

La classe GeometricSprite étend la classe flash.display.Sprite :

```
public class GeometricSprite extends Sprite
```

La classe GeometricSprite comprend diverses propriétés communes à tous les objets GeometricSprite. Elles sont définies dans la fonction constructeur en fonction des paramètres transmis à cette dernière. Par exemple :

```
this.size = size;
this.lineColor = lColor;
this.fillColor = fColor;
```

La propriété `geometricShape` de la classe GeometricSprite définit une interface IGeometricShape, qui stipule les propriétés mathématiques, mais non visuelles, de la forme. Les classes qui implémentent l'interface IGeometricShape sont définies dans l'exemple d'application GeometricShapes (voir « Exemple : GeometricShapes » à la page 126).

La classe GeometricSprite définit la méthode `drawShape()`, qui est affinée dans les définitions de remplacement de chaque sous-classe de GeometricSprite. Pour plus d'informations, consultez la section ci-après, Ajout d'objets d'affichage au rectangle de la zone de dessin.

La classe GeometricSprite propose également les méthodes suivantes :

- Les méthodes `onMouseDown()` et `onMouseUp()`, décrites à la section « Cliquer-déplacer un objet d'affichage » à la page 326

- Les méthodes `showSelected()` et `hideSelected()`, décrites à la section « Cliquer-déplacer un objet d'affichage » à la page 326

Ajout d'objets d'affichage au rectangle de la zone de dessin

Lorsque l'utilisateur clique sur le bouton Add Shape, l'application appelle la méthode `addShape()` de la classe DrawingCanvas. Elle crée une occurrence de GeometricSprite en appelant la fonction constructeur appropriée de l'une des sous-classes de GeometricSprite, comme illustré dans l'exemple suivant :

```
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);
}
```

Chaque méthode constructeur appelle la méthode `drawShape()`, qui utilise la propriété `graphics` de la classe (héritée de la classe Sprite) pour dessiner le graphique vectoriel approprié. Par exemple, la méthode `drawShape()` de la classe CircleSprite comprend le code suivant :

```
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);
```

L'avant-dernière ligne de la fonction `addShape()` définit la propriété `alpha` de l'objet d'affichage (héritée de la classe DisplayObject), de sorte que chaque objet d'affichage ajouté au rectangle de la zone de dessin soit légèrement transparent, permettant ainsi à l'utilisateur de visualiser les objets placés derrière.

La derrière ligne de la méthode `addChild()` ajoute le nouvel objet d'affichage à la liste enfant de l'occurrence de la classe DrawingCanvas, qui figure déjà dans la liste d'affichage. Le nouvel objet d'affichage apparaît alors sur la scène.

L'interface de l'application comprend deux champs de texte, `selectedSpriteTxt` et `outputTxt`. Les propriétés de texte de ces champs sont mises à jour avec des informations relatives aux objets GeometricSprite ajoutés au rectangle de la zone de dessin ou sélectionnés par l'utilisateur. La classe GeometricSprite gère ces tâches de transmission d'informations en annulant la méthode `toString()` comme suit :

```
public override function toString():String
{
    return this.shapeType + " of size " + this.size + " at " + this.x + ", " + this.y;
}
```

La propriété `shapeType` est définie sur la valeur appropriée de la méthode constructeur de chaque sous-classe GeometricSprite. La méthode `toString()` pourrait par exemple renvoyer la valeur suivante pour une occurrence de CircleSprite récemment ajoutée à l'occurrence de DrawingCanvas :

```
Circle of size 50 at 0, 0
```

La méthode `describeChildren()` de la classe DrawingCanvas parcourt la liste enfant du rectangle de la zone de dessin en utilisant la propriété `numChildren` (héritée de la classe DisplayObjectContainer) pour définir la limite de la boucle `for`. Elle génère une chaîne qui recense chaque enfant, comme suit :

```
var desc:String = "";
var child:DisplayObject;
for (var i:int=0; i < this.numChildren; i++)
{
child = this.getChildAt(i);
desc += i + ": " + child + '\n';
}
```

La chaîne résultante permet de définir la propriété `text` du champ de texte `outputTxt`.

Cliquer-déplacer un objet d'affichage

Lorsque l'utilisateur clique sur une occurrence de GeometricSprite, l'application appelle le gestionnaire de l'événement `onMouseDown()`. Comme le montre le code ci-dessous, ce gestionnaire écoute les événements de clic gauche de souris dans la fonction constructeur de la classe GeometricSprite :

```
this.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
```

La méthode `onMouseDown()` appelle ensuite la méthode `showSelected()` de l'objet GeometricSprite. S'il s'agit du premier appel de cette méthode sur l'objet, elle crée un objet Shape appelé `selectionIndicator` et utilise la propriété `graphics` de l'objet Shape pour dessiner un rectangle rouge de mise en valeur, comme suit :

```
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);
```

S'il ne s'agit pas du premier appel de la méthode `onMouseDown()`, celle-ci active simplement la propriété `visible` (héritée de la classe DisplayObject) de la forme `selectionIndicator` :

```
this.selectionIndicator.visible = true;
```

La méthode `hideSelected()` masque la forme `selectionIndicator` de l'objet précédemment sélectionné en définissant sa propriété `visible` sur `false`.

La méthode du gestionnaire d'événement `onMouseDown()` appelle également la méthode `startDrag()` (héritée de la classe Sprite), qui comprend le code suivant :

```
var boundsRect:Rectangle = this.parent.getRect(this.parent);
boundsRect.width -= this.size;
boundsRect.height -= this.size;
this.startDrag(false, boundsRect);
```

L'utilisateur peut alors déplacer l'objet sélectionné sur la zone de dessin, au sein des limites définies par le rectangle `boundsRect`.

Lorsque l'utilisateur relâche le bouton de la souris, l'événement `mouseUp` est distribué. La méthode constructeur de DrawingCanvas configure l'écouteur d'événement suivant :

```
this.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
```

Cet écouteur d'événement est associé à l'objet DrawingCanvas, plutôt qu'aux objets GeometricSprite individuels. En effet, lorsque l'utilisateur déplace l'objet GeometricSprite, celui-ci risque d'être placé derrière un autre objet d'affichage (un autre objet GeometricSprite) lorsque le bouton de la souris est relâché. L'événement « mouse up » s'appliquerait à l'objet d'affichage en avant-plan, mais non à l'objet d'affichage déplacé par l'utilisateur. Ajouter l'écouteur à l'objet DrawingCanvas assure la gestion de l'événement.

La méthode `onMouseUp()` appelle la méthode `onMouseUp()` de l'objet GeometricSprite, qui appelle alors la méthode `stopDrag()` de l'objet GeometricSprite.

Réorganisation de l'ordre de superposition des objets d'affichage

L'interface utilisateur de l'application comprend des boutons intitulés Move Back, Move Down, Move Up et Move to Front. Lorsque l'utilisateur clique sur l'un de ces boutons, l'application appelle la méthode correspondante de la classe DrawingCanvas, à savoir : `moveToBack()`, `moveDown()`, `moveUp()` ou `moveToFront()`. Par exemple, la méthode `moveToBack()` comporte le code suivant :

```
public function moveToBack(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, 0);
    }
}
```

Cette méthode utilise la méthode `setChildIndex()` (héritée de la classe DisplayObjectContainer) pour placer l'objet d'affichage à la position d'index 0 de la liste des enfants de l'occurrence de DrawingCanvas (`this`).

Le fonctionnement de la méthode `moveDown()` est similaire, mais elle décrémente la position d'index de l'objet d'affichage d'une unité dans la liste des enfants de l'occurrence de DrawingCanvas :

```
public function moveDown(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, index - 1);
    }
}
```

Le fonctionnement des méthodes `moveUp()` et `moveToFront()` est similaire aux méthodes `moveToBack()` et `moveDown()`.

Chapitre 14 : Utilisation de l'API de dessin

Bien que l'importance des images et graphiques importés soit évidente, il ne faut pas négliger la fonctionnalité connue sous le nom d'API de dessin, qui permet de tracer des lignes et des formes en ActionScript. Vous pouvez ainsi ouvrir une application avec l'équivalent informatique d'une toile vierge et y créer des images. La possibilité de créer des graphiques ouvre de nouvelles possibilités pour vos applications. Les techniques présentées dans ce chapitre permettent, entre autres, de créer un programme de dessin, de réaliser des éléments artistiques animés et interactifs, ou encore de générer par programmation vos propres éléments d'interface.

Principes de base de l'utilisation de l'API de dessin

Introduction à l'API de dessin

L'API de dessin est le nom des fonctionnalités intégrées à ActionScript qui permettent de créer des graphiques vectoriels (lignes, courbes, formes, remplissages et dégradés) et de les afficher à l'aide d'ActionScript. Ces fonctionnalités sont prises en charge par la classe flash.display.Graphics. ActionScript permet de dessiner dans une occurrence d'un objet de type Shape, Sprite ou MovieClip, à l'aide de la propriété `graphics` définie dans chacune de ces classes. (La propriété `graphics` de ces classes est en fait une occurrence de la classe Graphics.)

Si vous n'avez pas l'habitude de « dessiner » par code, la classe Graphics comprend plusieurs méthodes qui facilitent le tracé de formes courantes (cercles, ellipses, rectangles et rectangles à coins arrondis). Ces tracés peuvent être des lignes vides ou des formes remplies. Si vous avez besoin de fonctionnalités plus sophistiquées, la classe Graphics comporte aussi des méthodes destinées au tracé de lignes et de courbes de Bézier, qui peuvent être utilisées conjointement avec les fonctions trigonométriques de la classe Math pour créer n'importe quelle forme.

Flash Player 10 et Adobe AIR 1.5 prennent en charge une nouvelle API de dessin, qui vous permet de tracer intégralement des formes par programmation à l'aide d'une commande unique. Une fois que vous vous êtes familiarisé avec la classe Graphics et les tâches décrites à la section « Bases d'utilisation de l'API de dessin », passez à la section « Utilisation avancée de l'API de dessin » à la page 341 pour en savoir plus sur ces fonctions.

Tâches courantes faisant appel à l'API de dessin

Les opérations suivantes, qui sont décrites dans ce chapitre, peuvent être aisément accomplies dans ActionScript avec l'API de dessin:

- Définition de styles de ligne et de remplissage pour le traçage de formes
- Dessin de lignes droites et de courbes
- Utilisation de méthodes pour tracer des formes (cercles, ellipses et rectangles)
- Dessin avec des lignes et des remplissages en dégradé
- Définition d'une matrice de création de dégradé
- Utilisation de fonctions trigonométriques avec l'API de dessin
- Incorporation des fonctionnalités de l'API de dessin dans une animation

Concepts importants et terminologie

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Point d'ancrage : l'un des deux points d'extrémité d'une courbe de Bézier.
- Point de contrôle : point qui définit la direction et la forme d'une courbe de Bézier. Cette ligne courbe ne touche jamais le point de contrôle, mais elle s'arrondit comme si elle était tracée dans la direction de celui-ci.
- Espace de coordonnées : représentation graphique des coordonnées contenues dans un objet d'affichage, par rapport auquel sont positionnés les éléments enfant.
- Remplissage : partie intérieure opaque d'une forme constituée par le remplissage d'une ligne ou d'une forme ne possédant pas de ligne de contour.
- Dégradé : transition progressive d'une couleur à une ou plusieurs autres (par opposition à une couleur unie).
- Point : emplacement unique dans un espace de coordonnées. Dans le système de coordonnées en 2 dimensions utilisé dans ActionScript, un point est défini par son emplacement le long de l'axe x et de l'axe y (les coordonnées du point).
- Courbe de Bézier quadratique : type de courbe définie par une formule mathématique particulière. Dans ce type de courbe, la forme de la courbe est calculée à partir des positions des points d'ancrage (les points d'extrémité de la courbe) et d'un point de contrôle qui définit la forme et la direction de la courbe.
- Echelle : taille relative d'un objet par rapport à sa taille d'origine. Mettre un objet à l'échelle consiste à modifier sa taille en l'étirant ou en le rétrécissant.
- Trait : ligne de contour d'une forme constituée par le remplissage de cette ligne, ou forme ne possédant pas de remplissage.
- Translation : modifier les coordonnées d'un point d'un espace de coordonnées à un autre.
- Axe X : axe horizontal dans le système de coordonnées en 2 dimensions utilisé dans ActionScript.
- Axe Y : axe vertical dans le système de coordonnées en 2 dimensions utilisé dans ActionScript.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester des exemples de code. Etant donné que ce chapitre décrit comment tracer du contenu visuel, le test des exemples de code implique l'exécution du code et l'affichage des résultats dans le fichier SWF créé. Pour tester les exemples de code :

1. Créez un document Flash vide.
2. Sélectionnez une image-clé dans le scénario.
3. Ouvrez le panneau Actions et copiez le code dans le panneau Script.
4. Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

   Les résultats des exemples de code apparaîtront dans le fichier SWF créé.

Présentation de la classe Graphics

Chaque objet Shape, Sprite et MovieClip possède une propriété `graphics` qui est une occurrence de la classe Graphics. La classe Graphics comporte des propriétés et des méthodes permettant de tracer des lignes, des remplissages et des formes. Pour disposer d'un objet d'affichage qui sera uniquement utilisé comme « toile de fond » pour un dessin, utilisez une occurrence de Shape. Une occurrence de Shape est mieux adaptée au dessin que les autres objets, car elle ne comporte pas les fonctionnalités (inutiles dans ce type d'utilisation) des classes Sprite et MovieClip. Par contre, si vous souhaitez créer un objet d'affichage qui servira de support à du contenu graphique mais doit également pouvoir contenir d'autres objets d'affichage, utilisez une occurrence de Sprite. Pour plus d'informations sur le choix des objets d'affichage en fonction de la tâche prévue, consultez la section « Sélection d'une sous-classe de DisplayObject » à la page 298

Dessin de lignes et de courbes

Tous les graphiques qu'il est possible de réaliser à l'aide d'une occurrence de la classe Graphics sont des tracés à l'aide de lignes et de courbes. Tous les dessins réalisés en ActionScript doivent donc suivre les mêmes étapes :

- Définition de styles de ligne et de remplissage
- Définition de la position de dessin initiale
- Dessin de lignes, courbes et formes (éventuellement en déplaçant le point de traçage)
- Le cas échéant, création d'un remplissage

Définition de styles de ligne et de remplissage

Pour utiliser la propriété `graphics` d'une occurrence de Shape, Sprite ou MovieClip, vous devez d'abord définir le style (épaisseur et couleur de la ligne, couleur de remplissage) à utiliser. De la même manière qu'avec les outils de dessin d'Adobe® Flash® CS4 Professional ou de toute autre application de dessin, en ActionScript vous pouvez dessiner avec ou sans trait, et avec ou sans remplissage. Pour indiquer l'apparence du trait, utilisez la méthode `lineStyle()` ou `lineGradientStyle()`. Pour créer une ligne pleine, utilisez la méthode `lineStyle()`. Lors de l'appel de cette méthode, les valeurs les plus courantes à indiquer sont les trois premiers paramètres : épaisseur de ligne, couleur et alpha. Par exemple, la ligne de code suivante indique que la forme `myShape` doit tracer des lignes de deux pixels d'épaisseur, en rouge (0x990000) et avec une opacité de 75 %:

```
myShape.graphics.lineStyle(2, 0x990000, .75);
```

La valeur par défaut du paramètre alpha est 1.0 (100 %), vous pouvez donc l'omettre si vous souhaitez tracer une ligne entièrement opaque. La méthode `lineStyle()` accepte également deux paramètres supplémentaires pour l'indice de lissage des pixels et le mode de mise à l'échelle. Pour plus d'informations sur ces paramètres, consultez la description de la méthode `Graphics.lineStyle()` dans le Guide de référence du langage et des composants ActionScript 3.0.

Pour créer une ligne dégradée, utilisez la méthode `lineGradientStyle()`. Pour plus d'informations sur cette méthode, consultez la section « Création de lignes et de remplissages en dégradé » à la page 333.

Pour créer une forme remplie, appelez les méthodes `beginFill()`, `beginGradientFill()`, `beginBitmapFill()` ou `beginShaderFill()` avant de débuter le dessin. La plus basique, `beginFill()`, accepte deux paramètres : la couleur de remplissage et, le cas échéant, la valeur alpha correspondante. Par exemple, pour tracer une forme avec un remplissage vert uni, utilisez le code suivant (on suppose ici que vous dessinez dans un objet nommé `myShape`) :

```
myShape.graphics.beginFill(0x00FF00);
```

L'appel d'une méthode de remplissage annule implicitement le remplissage précédemment défini avant l'implémentation du nouveau. L'appel d'une méthode qui spécifie un style de trait remplace le style de trait précédent, mais ne modifie pas le remplissage précédemment défini, et vice-versa.

Une fois spécifié le style de ligne et de remplissage, l'étape suivante consiste à indiquer le point de départ du dessin. L'occurrence de Graphics possède un point de traçage, tout comme la pointe d'un crayon sur une feuille de papier. Quel que soit l'emplacement du point de traçage, il représente l'origine de l'action de dessin à venir. Initialement, un objet Graphics débute avec son point de traçage aux points 0,0 dans l'espace de coordonnées de l'objet dans lequel il dessine. Pour que le tracé débute en un autre point, appelez la méthode `moveTo()` avant d'appeler une des méthodes de dessin. Cet appel peut être comparé à l'action de lever la pointe du crayon du papier et de l'amener à un nouvel emplacement.

Lorsque le point de traçage est en place, utilisez une série d'appels aux méthodes `lineTo()` (pour tracer des lignes droites) et `curveTo()` (pour tracer des courbes).

Pendant l'opération de dessin, vous pouvez à tout moment appeler la méthode `moveTo()` pour amener le point de traçage à une nouvelle position sans dessiner.

Pendant le traçage du dessin, si vous avez indiqué une couleur de remplissage, vous pouvez ordonner à Adobe Flash Player or Adobe® AIR™ de fermer le remplissage en appelant la méthode `endFill()`. Si vous n'avez pas tracé une forme close (autrement dit, si lors de l'appel de la méthode `endFill()` le point de traçage n'est pas sur les coordonnées du point de départ de la forme), lorsque vous appelez la méthode `endFill()`, Flash Player ou AIR ferme automatiquement la forme en traçant une ligne droite entre le point de traçage actuel et l'emplacement spécifié dans le dernier appel à la méthode `moveTo()`. Si vous avez débuté un remplissage et n'avez pas appelé `endFill()`, tout appel à `beginFill()` (ou à l'une des autres méthodes de remplissage) ferme le remplissage actif et en débute un nouveau.

Dessin de lignes droites

Lorsque vous appelez la méthode `lineTo()`, l'objet Graphics trace une ligne droite (en utilisant le style de ligne que vous avez spécifié) entre le point de traçage actuel et les coordonnées que vous transmettez en paramètres à cette méthode. Par exemple, cette ligne de code place le point de traçage aux coordonnées 100, 100 puis trace une ligne jusqu'au point 200, 200 :

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.lineTo(200, 200);
```

L'exemple suivant trace des triangles rouges et verts d'une hauteur de 100 pixels :

```
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);
```

Dessin de courbes

La méthode `curveTo()` dessine une courbe de Bézier. Elle trace un arc entre deux points (appelés points d'ancrage) courbé en direction d'un troisième point (appelé point de contrôle). L'objet Graphics utilise la position de traçage actuelle comme premier point d'ancrage. Lorsque vous appelez la méthode `curveTo()`, vous transmettez quatre paramètres : les coordonnées x et y du point de contrôle, puis les coordonnées x et y du second point d'ancrage. Par exemple, le code suivant trace une courbe entre le point 100, 100 et le point 200, 200. Le point de contrôle ayant les coordonnées 175, 125, la courbe est orientée vers la droite puis vers le bas :

```
myShape.graphics.moveTo(100, 100);
myShape.graphics.curveTo(175, 125, 200, 200);
```

L'exemple suivant trace des objets circulaires rouges et verts avec une largeur et une hauteur de 100 pixels. Notez qu'en raison même de la nature de l'équation de Bézier, ces cercles ne sont pas parfaits:

```
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);
```

Dessin de formes à l'aide des méthodes intégrées

Pour vous permettre de tracer plus commodément des formes courantes (cercles, ellipses, rectangles et rectangles à coins arrondis), ActionScript 3.0 comporte des méthodes qui tracent automatiquement ces formes. Ces méthodes sont `drawCircle()`, `drawEllipse()`, `drawRect()`, `drawRoundRect()` et `drawRoundRectComplex()`, et sont toutes définies dans la classe Graphics. Elles peuvent être utilisées à la place des méthodes `lineTo()` et `curveTo()`. Notez toutefois qu'il est nécessaire de spécifier des styles de ligne et de remplissage avant d'appeler ces méthodes.

L'exemple suivant dessine des carrés bleus, rouges et verts avec une largeur et une hauteur de 100 pixels. Ce code utilise la méthode `drawRect()` et spécifie que l'opacité de la couleur de remplissage est de 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);
```

Dans un objet Sprite ou MovieClip, tout contenu graphique créé à l'aide de la propriété `graphics` apparaît toujours derrière les objets d'affichage enfant contenus par cet objet. Par ailleurs, le contenu créé avec la propriété `graphics` n'est pas un objet d'affichage séparé. Il n'apparaît donc pas dans la liste des enfants d'un objet Sprite ou MovieClip. Par exemple, l'objet Sprite suivant reçoit l'instruction de tracer un cercle avec sa propriété `graphics`, et un objet TextField figure dans sa liste d'objets d'affichage enfant :

```
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);
```

Notez que l'objet TextField apparaît au-dessus du cercle tracé avec la propriété graphics.

Création de lignes et de remplissages en dégradé

L'objet Graphics permet aussi de tracer des traits et des remplissages avec des dégradés au lieu de couleurs unies. Pour créer un trait dégradé, utilisez la méthode `lineGradientStyle()`. Pour créer un remplissage dégradé, utilisez la méthode `beginGradientFill()`.

Ces deux méthodes reçoivent les mêmes paramètres. Les quatre premiers sont obligatoires : type, couleurs, transparences alpha et rapports. Les quatre suivants sont facultatifs mais peuvent être utiles pour plus de personnalisation.

- Le premier paramètre spécifie le type de dégradé à créer. Les valeurs acceptables sont `GradientFill.LINEAR` et `GradientFill.RADIAL`.
- Le deuxième paramètre indique le tableau de valeurs colorimétriques à utiliser. Dans un dégradé linéaire, les couleurs sont organisées de gauche à droite. Dans un dégradé radial, les couleurs sont organisées de l'intérieur à l'extérieur. L'ordre des couleurs dans le tableau représente l'ordre dans lequel elles sont tracées dans le dégradé.
- Le troisième paramètre indique les valeurs de transparence alpha pour les couleurs correspondantes du paramètre précédent.
- Le quatrième paramètre spécifie les rapports, c'est-à-dire l'importance de chaque couleur dans le dégradé. Les valeurs acceptables vont de 0 à 255. Ces valeurs ne représentent pas une hauteur ou une largeur, mais plutôt la position au sein du dégradé : 0 représente le début du dégradé, et 255 la fin. Cette plage de rapports doit augmenter séquentiellement et comporter le même nombre d'éléments que les tableaux des couleurs et des valeurs alpha spécifiés comme deuxième et troisième paramètres.

Le cinquième paramètre, la matrice de transformation, est facultatif mais fréquemment utilisé, car il représente un moyen à la fois puissant et aisé de contrôler l'aspect du dégradé. Ce paramètre accepte une occurrence de l'objet Matrix. Le moyen le plus simple de créer un objet Matrix pour un dégradé consiste à utiliser la méthode `createGradientBox()` de la classe Matrix.

Définition d'un objet Matrix à utiliser avec un dégradé

Utilisez les méthodes `beginGradientFill()` et `lineGradientStyle()` de la classe flash.display.Graphics pour définir les dégradés à utiliser dans des formes. Lorsque vous définissez un dégradé, vous fournissez une matrice comme l'un des paramètres de ces méthodes.

La façon la plus facile de définir la matrice est d'utiliser la méthode `createGradientBox()`, de la classe Matrix, qui définit un tableau utilisé pour définir le dégradé. Définissez l'échelle, la rotation et la position du dégradé à l'aide des paramètres transmis à la méthode `createGradientBox()`. La méthode `createGradientBox()` reçoit les paramètres suivants :

- Largeur de la zone de dégradé : largeur (en pixels) sur laquelle s'étend le dégradé
- Hauteur de la zone de dégradé : hauteur (en pixels) sur laquelle s'étend le dégradé
- Rotation de la zone de dégradé : rotation (en radians) qui sera appliquée au dégradé
- Translation horizontale : distance (en pixels) de déplacement horizontal du dégradé
- Translation verticale : distance (en pixels) de déplacement vertical du dégradé

Par exemple, supposons un dégradé possédant les caractéristiques suivantes :

- `GradientType.LINEAR`
- Deux couleurs, vert et bleu, avec le tableau `ratios` défini sur `[0, 255]`
- `SpreadMethod.PAD`
- `InterpolationMethod.LINEAR_RGB`

Les exemples suivants présentent des dégradés dans lesquels le paramètre `rotation` de la méthode `createGradientBox()` varie comme indiqué, mais tous les autres réglages restent inchangés :

`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;`	

Les exemples suivants présentent les effets sur un dégradé linéaire vert à bleu dans lequel les paramètres `rotation`, `tx` et `ty` de la méthode `createGradientBox()` varient comme indiqué, mais tous les autres réglages restent inchangés:

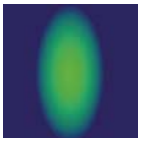
`width = 50;`
`height = 100;`
`rotation = 0;`
`tx = 0;`
`ty = 0;`	

`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;`	

Les paramètres `width`, `height`, `tx` et `ty` de la méthode `createGradientBox()` ont une incidence sur la taille et la position d'un remplissage en dégradé *radial* également, comme indiqué dans l'exemple suivant :

`width = 50;`
`height = 100;`
`rotation = 0;`
`tx = 25;`
`ty = 0;`
	

Le code suivant produit le dernier dégradé radial illustré :

```
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);
```

Notez que la largeur et la hauteur du dégradé sont déterminées par la largeur et la hauteur de la matrice du dégradé, plutôt que par celles qui sont dessinées à l'aide de l'objet Graphics. Si vous dessinez avec l'objet Graphics, vous tracez ce qui existe à ces coordonnées dans la matrice du dégradé. Même si vous utilisez l'une des méthodes de création de forme d'un objet de type Graphics, par exemple `drawRect()`, le dégradé n'est pas étiré en fonction de la taille de la forme dessinée : sa taille doit être spécifiée dans la matrice du dégradé.

L'exemple ci-dessous illustre la différence visuelle entre les dimensions de la matrice du dégradé et celles du dessin :

```
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);
```

Ce code trace trois dégradés avec le même style de remplissage, spécifié avec une distribution égale de rouge, vert et bleu. Les dégradés sont tracés à l'aide de la méthode `drawRect()` avec des largeurs en pixels de 50, 100 et 150 respectivement. La matrice de dégradé qui est spécifiée dans la méthode `beginGradientFill()` est créée avec une largeur de 100 pixels. Le premier dégradé ne couvre donc que la moitié de son spectre, le deuxième le couvre en entier, et le troisième le couvre en entier et possède 50 pixels supplémentaires de bleu à droite.

La méthode `lineGradientStyle()` fonctionne de façon similaire à `beginGradientFill()`, si ce n'est qu'outre la définition du dégradé vous devez aussi indiquer l'épaisseur du trait à l'aide de la méthode `lineStyle()` avant de tracer. Le code suivant trace une boîte avec un trait dégradé rouge, vert et bleu :

```
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);
```

Pour plus d'informations sur la classe Matrix, consultez la section « Utilisation des objets Matrix » à la page 357.

Utilisation de la classe Math avec les méthodes de dessin

Un objet Graphics trace des cercles et des carrés, mais il permet aussi de dessiner des formes plus complexes, en particulier lorsque les méthodes de dessin sont utilisées en combinaison avec les propriétés et les méthodes de la classe Math. La classe Math contient des constantes d'intérêt général en mathématiques, telles que `Math.PI` (environ 3,14159265...), qui est la constante définissant le rapport entre la circonférence et le diamètre d'un cercle. Elle contient également des méthodes de fonctions trigonométriques, entre autres `Math.sin()`, `Math.cos()` et `Math.tan()`. L'utilisation de ces méthodes et constantes pour le dessin de formes permet d'obtenir des effets visuels plus dynamiques, en particulier en utilisant des répétitions et des récursions.

De nombreuses méthodes de la classe Math attendent des mesures circulaires en radians, et non en degrés. La conversion entre ces deux types d'unités représente elle-même un cas d'utilisation courante de la classe Math :

```
var degrees = 121;
var radians = degrees * Math.PI / 180;
trace(radians) // 2.111848394913139
```

L'exemple suivant crée une onde sinusoïdale et une onde cosinusoïdale, afin d'illustrer la différence entre les méthodes `Math.sin()` et `Math.cos()` pour une même valeur.

```
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);
}
```

Animation avec l'API de dessin

L'un des avantages de la création de contenu graphique avec l'API de dessin est que ce contenu peut être repositionné à loisir. Tout ce que vous tracez peut être modifié, en modifiant simplement les variables utilisées pour ce dessin. Vous pouvez donc obtenir de l'animation en changeant les variables et en retraçant, soit sur un nombre d'images donné, soit à l'aide d'un timer.

Par exemple, le code suivant change l'affichage à chaque nouvelle image (en écoutant l'événement `Event.ENTER_FRAME`) : il incrémente la valeur de degrés actuelle, puis ordonne à l'objet graphique d'effacer et redessiner avec la nouvelle 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;
}
```

Pour produire un résultat nettement différent, vous pouvez expérimenter en modifiant les valeurs initiales au début du code, `currentDegrees`, `radius` et `satelliteRadius`. Par exemple, essayez en réduisant la valeur de la variable radius et/ou en augmentant la valeur de la variable totalSatellites. Ce n'est qu'un exemple simple de la façon dont l'API de dessin permet de créer du contenu visuel dont la complexité dissimule la simplicité de création.

Exemple : générateur algorithmique d'effets visuels

L'exemple de générateur algorithmique d'effets visuels trace de manière dynamique plusieurs « satellites », des cercles qui se déplacent suivant une orbite circulaire. Les fonctionnalités présentées sont les suivantes :

- Utilisation de l'API de dessin pour tracer une forme simple avec un aspect dynamique
- Utilisation de l'interaction de l'utilisateur pour modifier les propriétés utilisées pour le dessin
- Effet d'animation par effacement du contenu et retraçage à chaque nouvelle image.

L'exemple de la section précédente animait un satellite isolé à l'aide de l'événement `Event.ENTER_FRAME`. Cet exemple le reprend en y ajoutant un panneau de contrôle avec divers curseurs qui actualisent immédiatement l'affichage de plusieurs satellites. Dans cet exemple, le code est formalisé dans des classes externes et le code de création du satellite est imbriqué dans une boucle, en conservant une référence à chaque satellite dans le tableau `satellites`.

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers de l'application se trouvent dans le dossier Samples/AlgorithmicVisualGenerator. Celui-ci contient les fichiers suivants :

Fichier	Description
AlgorithmicVisualGenerator.fla	Fichier d'application principal FLA pour Flash.
com/example/programmingas3/algorithmic/AlgorithmicVisualGenerator.as	Classe comportant les principales fonctionnalités de l'application : dessin des satellites sur la scène et actualisation des variables utilisées pour ces dessins en réponse aux événements du panneau de contrôle.
com/example/programmingas3/algorithmic/ControlPanel.as	Cette classe gère les interactions de l'utilisateur avec les curseurs et distribue les événements appropriés.
com/example/programmingas3/algorithmic/Satellite.as	Cette classe représente l'objet d'affichage qui tourne sur son orbite autour d'un point central et contient des propriétés relatives à son état de dessin actuel.

Définition des écouteurs

L'application commence par créer trois écouteurs. Le premier attend qu'un événement soit distribué par le panneau de contrôle pour signaler qu'une reconstruction des satellites est nécessaire. Le second attend des changements de taille de la scène du fichier SWF. Le troisième attend le passage de chaque image du fichier SWF et son retraçage à l'aide de la fonction `doEveryFrame()`.

Création des satellites

Après la définition de ces écouteurs, la fonction `build()` est appelée. Cette fonction appelle d'abord la fonction `clear()`, qui efface le contenu du tableau `satellites` et supprime les éventuelles formes déjà présentes sur la scène. Cette précaution est nécessaire car la fonction `build()` peut être appelée à nouveau à la suite d'un événement diffusé par le panneau de contrôle, par exemple en cas de modification des couleurs. Dans ce cas, les satellites doivent être supprimés et recréés.

La fonction crée ensuite les satellites, en définissant les propriétés initiales nécessaires à cette création, dont la variable `position` qui définit une position aléatoire sur l'orbite et la variable `color`, qui dans cet exemple ne change pas une fois que le satellite a été créé.

Lors de la création de chaque satellite, une référence est ajoutée dans le tableau `satellites`. Lorsque la fonction `doEveryFrame()` est appelée, elle actualise tous les satellites du tableau.

Actualisation de la position des satellites

La fonction `doEveryFrame()` est au cœur du processus d'animation de l'application. Elle est appelée à chaque image, donc à une fréquence identique à la cadence du fichier SWF. Les variables du dessin changent légèrement, ce qui permet d'obtenir un effet d'animation.

La fonction efface d'abord tous les éventuels dessins antérieurs et retrace l'arrière-plan. Elle effectue ensuite une boucle dans le conteneur de chaque satellite et incrémente la propriété `position` de chaque satellite, puis actualise les propriétés `radius` et `orbitRadius` qui peuvent avoir été modifiées par suite d'une action de l'utilisateur dans le panneau de contrôle. Enfin, la nouvelle position de chaque satellite est affichée en appelant la méthode `draw()` de la classe Satellite.

Notez que la variable de compteur `i` n'est incrémentée que jusqu'à la valeur de la variable `visibleSatellites`. En effet, si l'utilisateur a limité à l'aide du panneau de contrôle le nombre de satellites affichés, les autres satellites de la boucle ne doivent pas être redessinés, mais masqués. La boucle chargée de cette action suit immédiatement celle qui est responsable du dessin.

Lorsque la fonction doEveryFrame() se termine, le nombre de satellites visibles (`visibleSatellites`) est redessiné aux nouvelles positions.

Réponse aux interactions de l'utilisateur

L'interactivité est assurée par le panneau de contrôle, qui est géré par la classe ControlPanel. Cette classe définit pour chaque curseur un écouteur et des valeurs individuelles minimum, maximum et par défaut. Lorsque l'utilisateur déplace ces curseurs, la fonction `changeSetting()` est appelée. Elle actualise les propriétés du panneau de contrôle. Si la modification nécessite un nouvel affichage, un événement est distribué et pris en charge dans le fichier principal de l'application. En fonction de la modification signalée par le panneau de contrôle, la fonction `doEveryFrame()` redessine chaque satellite sur la base des nouvelles variables.

Améliorations possibles

Cet exemple est simplement destiné à illustrer la création d'effets visuels avec l'API de dessin. Il utilise relativement peu de lignes de code pour créer une animation interactive qui semble très complexe. Même ainsi, cet exemple pourrait être amélioré moyennant quelques modifications mineures. Voici quelques idées :

- La fonction `doEveryFrame()` pourrait incrémenter la valeur colorimétrique du satellite.
- La fonction `doEveryFrame()` pourrait réduire ou augmenter progressivement le rayon de l'orbite du satellite.
- Il n'est pas nécessaire que l'orbite du satellite soit circulaire, la classe Math permet de le déplacer selon une sinusoïdale, par exemple.
- Les satellites pourraient utiliser une détection de collision entre eux.

L'API de dessin peut être considérée comme autre solution pour la création d'effets visuels dans l'environnement de programmation Flash, en dessinant des formes de base lors de l'exécution. Mais elle permet aussi de créer des effets visuels qu'il serait impossible de créer manuellement. L'API de dessin et quelques notions de mathématiques permettent au développeur en ActionScript de donner vie à de nombreuses créations inattendues.

Utilisation avancée de l'API de dessin

Introduction à l'utilisation avancée de l'API de dessin

Flash Player 10 et Adobe AIR 1.5 prennent désormais en charge un ensemble de fonctions de dessin avancées. Les améliorations de l'API de dessin, qui étendent les méthodes de dessin des versions antérieures, vous permettent de définir des ensembles de données pour générer des formes, en modifier lors de l'exécution et créer des effets tridimensionnels. Les améliorations de l'API de dessin consolident les méthodes existantes comme commandes alternatives. Ces commandes s'appuient sur des tableaux de vecteurs et des classes d'énumération pour fournir des ensembles de données aux méthodes de dessin. Les tableaux de vecteurs accélèrent le rendu de formes plus complexes. Les développeurs peuvent modifier les valeurs des tableaux par programmation pour rendre des formes dynamiques à l'exécution.

Les nouvelles fonctions de dessins de Flash Player 10 sont décrites dans les sections suivantes : « Tracés de dessin » à la page 342, « Définition des règles d'enroulement » à la page 344, « Utilisation des classes de données graphiques » à la page 346 et « A propos de l'utilisation de drawTriangles() » à la page 349.

Tâches courantes faisant appel à l'API de dessin avancée

Vous souhaiterez probablement effectuer les tâches suivantes à l'aide des fonctions avancées de l'API de dessin dans ActionScript :

- Stockage des données destinées aux méthodes de dessin à l'aide d'objets Vector
- Définition de tracés pour tracer des formes par programmation
- Définition de règles d'enroulement pour déterminer le remplissage de formes se chevauchant
- Utilisation des classes de données graphiques
- Utilisation de triangles et de méthodes de dessin pour obtenir des effets tridimensionnels.

Concepts importants et terminologie

La liste de référence suivante énumère les termes importants que vous rencontrerez dans cette section :

- Vecteur : tableau de valeurs d'un type de données identique. Un objet Vector peut stocker un tableau de valeurs qu'utilisent des méthodes de dessin pour construire des lignes et des formes par le biais d'une commande unique. Pour plus d'informations sur les objets Vector, consultez la section « Tableaux indexés » à la page 160.
- Tracé : un tracé est composé d'un ou de plusieurs segments droits ou incurvés. Le début et la fin de chaque segment sont indiqués par des coordonnées qui fonctionnent à la manière d'épingles maintenant un fil en place. Un tracé peut être fermé (un cercle, par exemple) ou ouvert, s'il comporte des extrémités distinctes (une ligne onduleuse, par exemple).
- Enroulement : direction d'un tracé tel qu'il est interprété par le rendu, soit positive (sens horaire) soit négative (sens antihoraire).
- GraphicsStroke : classe permettant de définir le style de ligne. Bien que le « trait » à proprement parler n'ait pas été inclus dans la nouvelle API de dessin, l'utilisation d'une classe pour désigner un style de ligne avec ses propres propriétés de remplissage constitue l'une des améliorations intégrées. Vous pouvez régler dynamiquement le style d'une ligne à l'aide de la classe GraphicsStroke.
- Objet Fill : objet créé à l'aide de classes d'affichage telles que flash.display.GraphicsBitmapFill et flash.display.GraphicsGradientFill, et transmis à la commande de dessin `Graphics.drawGraphicsData()`. Les objets Fill et les commandes de dessin optimisées proposent une approche de programmation plus orientée objets pour répliquer `Graphics.beginBitmapFill()` et `Graphics.beginGradientFill()`.

Tracés de dessin

La section sur le dessin de lignes et de courbes (consultez « Dessin de lignes et de courbes » à la page 330) a présenté les commandes permettant de tracer une ligne (`Graphics.lineTo()`) ou courbe (`Graphics.curveTo()`) unique, puis de la déplacer vers un autre point (`Graphics.moveTo()`) pour obtenir une forme. Flash Player 10 et Adobe AIR 1.5 prennent désormais en charge les nouvelles fonctions de l'API de dessin ActionScript, telles que `Graphics.drawPath()` et `Graphics.drawTriangles()`, qui utilisent les commandes de dessin existantes comme paramètres. Une série de commandes `Graphics.lineTo()`, `Graphics.curveTo()` ou `Graphics.moveTo()` est donc exécutée sous la forme d'une instruction unique.

Deux des nouvelles fonctions de l'API de dessin permettent à `Graphics.drawPath()` et à `Graphics.drawTriangles()` de consolider les commandes existantes :

- Classe d'énumération GraphicsPathCommand : cette classe associe plusieurs commandes de dessin à des valeurs constantes. Vous utilisez une série de ces valeurs comme paramètres de la méthode `Graphics.drawPath()`. Vous pouvez ensuite rendre une forme entière ou plusieurs formes à l'aide d'une seule commande. Vous pouvez aussi modifier dynamiquement les valeurs transmises à ces méthodes pour modifier une forme existante.
- Tableaux de vecteurs : les tableaux de vecteurs contiennent une série de valeurs d'un type de données spécifique. Vous stockez une série de constantes GraphicsPathCommand dans un objet Vector et une série de coordonnées dans un autre objet Vector. `Graphics.drawPath()` ou `Graphics.drawTriangles()` attribue ces valeurs ensemble pour générer un tracé de dessin ou une forme.

Il n'est plus nécessaire d'utiliser des commandes distinctes pour chaque segment d'une forme. Par exemple, la méthode `Graphics.drawPath()` conjugue `Graphics.moveTo()`, `Graphics.lineTo()` et `Graphics.curveTo()`. Au lieu d'être appelées séparément, ces méthodes sont converties en identifiants numériques tels que définis dans la classe GraphicsPathCommand. Une opération `moveTo()` est représentée par un `1`, une opération `lineTo()` par un `2`. Stockez un tableau de ces valeurs dans un objet Vector.<int> à utiliser dans le paramètre `commands`. Créez ensuite un autre tableau contenant des coordonnées dans un objet Vector.<Number> à utiliser dans le paramètre `data`. Chaque valeur GraphicsPathCommand correspond aux valeurs de coordonnée stockées dans le paramètre data, deux nombres consécutifs définissant un emplacement dans l'espace de coordonnées cible.

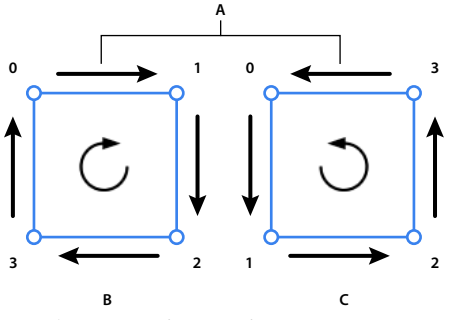
Remarque : *les valeurs du vecteur ne sont pas des objets Point. Le vecteur est une série de nombres, dont chaque paire représente une paire de coordonnées x/y.*

La méthode `Graphics.drawPath()` fait correspondre chaque commande à ses valeurs de point respectives (une série de deux ou quatre nombres) pour générer un tracé dans l'objet Graphics :

```
package{
import flash.display.*;

public class DrawPathExample extends Sprite {
    public function DrawPathExample(){

    var square_commands:Vector.<int> = new Vector.<int>(5,true);
    square_commands[0] = 1;//moveTo
    square_commands[1] = 2;//lineTo
    square_commands[2] = 2;
    square_commands[3] = 2;
    square_commands[4] = 2;

    var square_coord:Vector.<Number> = new Vector.<Number>(10,true);
    square_coord[0] = 20; //x
    square_coord[1] = 10; //y
    square_coord[2] = 50;
    square_coord[3] = 10;
    square_coord[4] = 50;
    square_coord[5] = 40;
    square_coord[6] = 20;
    square_coord[7] = 40;
    square_coord[8] = 20;
    square_coord[9] = 10;

    graphics.beginFill(0x442266);//set the color
    graphics.drawPath(square_commands, square_coord);
    }
}
}
```

Définition des règles d'enroulement

Flash Player 10 et Adobe AIR 1.5 introduisent également le concept « d'enroulement » de tracé, c'est-à-dire la direction d'un tracé. L'enroulement d'un tracé est soit positif (sens horaire) soit négatif (sens antihoraire). L'ordre dans lequel le rendu interprète les coordonnées que fournit le vecteur au paramètre data détermine l'enroulement.



Enroulement positif et négatif
A. *Les flèches indiquent la direction du tracé.* ***B.*** *Enroulement positif (sens horaire)* ***C.*** *Enroulement négatif (sens antihoraire)*

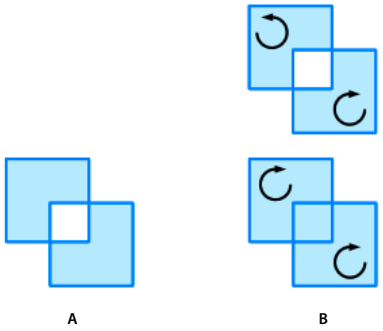
En outre, vous remarquerez que la méthode `Graphics.drawPath()` possède un troisième paramètre facultatif, appelé « winding » :

```
drawPath(commands:Vector.<int>, data:Vector.<Number>, winding:String = "evenOdd"):void
```

Dans ce contexte, le troisième paramètre est une chaîne ou une constante qui détermine la règle d'enroulement ou de remplissage de tracés se croisant : (Les valeurs de constante sont définies dans la classe GraphicsPathWinding sous la forme `GraphicsPathWinding.EVEN_ODD` ou `GraphicsPathWinding.NON_ZERO`.) La règle d'enroulement est importante lorsque des tracés se croisent.

Règle d'enroulement standard, la règle pair-impair est utilisée par l'API de dessin héritée. C'est aussi la règle par défaut de la méthode `Graphics.drawPath()`. Lorsqu'elle est appliquée, des tracés qui se croisent alternent entre des remplissages ouverts et fermés. Si deux carrés utilisant un même remplissage se croisent, la zone d'intersection est remplie. En règle générale, les zones adjacentes ne sont pas remplies ou leur remplissage n'est pas effacé.

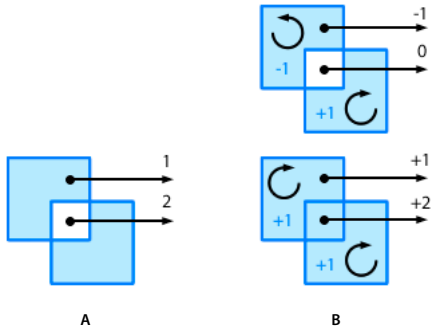
La règle non null, en revanche, se fonde sur l'enroulement (direction du tracé) pour déterminer si les zones définies par des tracés qui se croisent sont remplies. Lorsque des tracés dont l'enroulement est différent se croisent, la zone définie n'est pas remplie, comme avec la règle pair-impair. Si l'enroulement est identique, la zone dont le remplissage serait effacé est remplie :



Règles d'enroulement associée aux zones d'intersection
A. *Règle d'enroulement pair-impair* ***B.*** *Règle d'enroulement non null*

Nom des règles d'enroulement

Les noms se réfèrent à une règle beaucoup plus spécifique qui définit comment ces remplissages sont gérés. On affecte aux chemins d'enroulement positifs une valeur +1 et aux négatifs une valeur -1. A partir d'un point au sein d'une surface close d'une forme, tirez un trait qui s'étend indéfiniment. Le nombre de fois que cette ligne traverse un chemin, ainsi que les valeurs combinées de ces chemins, est utilisé pour déterminer le remplissage. Pour des enroulements pair-impair, c'est le nombre de fois que la ligne traverse un chemin qui est utilisé. Lorsque le décompte est un nombre impair, la zone est remplie. Lorsqu'il est impair, la zone ne l'est pas. Pour des enroulements non nuls, ce sont les valeurs affectées aux chemins qui sont utilisées. Lorsque les valeurs combinées du chemin ne sont pas nulles, la zone est remplie. Lorsque la valeur combinée est 0, la zone n'est pas remplie.



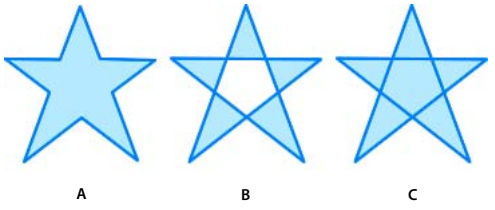
Règles de décompte et remplissage d'enroulements
A. *Règle d'enroulement pair-impair* ***B.*** *Règle d'enroulement non nul*

Utilisation des règles d'enroulement

Ces règles de remplissage sont complexes mais elles s'avèrent nécessaires dans certaines situations. Par exemple, prenons le dessin d'une forme étoilée. Suivant la règle pair-impair standard, la forme nécessiterait dix lignes différentes. Suivant la règle d'enroulement non nul, ces dix lignes sont réduites à cinq. Voici le code ActionScript pour une étoile à cinq lignes et une règle d'enroulement non nul :

```
fill.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);
```

Et voici la forme de l'étoile :



Une forme d'étoile qui utilise différentes règles d'enroulement
A. *10 lignes en pair-impair* ***B.*** *5 lignes en pair-impair* ***C.*** *5 lignes non nulles*

Et, comme les images sont animées ou utilisées comme textures sur des objets à trois dimensions et qu'elles se recouvrent, les règles d'enroulement deviennent plus importantes.

Utilisation des classes de données graphiques

Flash Player 10 et Adobe AIR 1.5 contiennent un nouvel ensemble de classes, résidant dans le package flash.display, du type IGraphicsData (une interface que chaque classe implémente). Les classes qui implémentent l'interface IGraphicsData servent de conteneurs de données aux méthodes de l'API de dessin.

Les classes suivantes implémentent l'interface IGraphicsData :

- GraphicsBitmapFill
- GraphicsEndFill
- GraphicsGradientFill
- GraphicsPath

- GraphicsShaderFill
- GraphicsSolidFill
- GraphicsStroke
- GraphicsTrianglePath

Ces classes vous permettent de stocker des dessins complets dans un tableau d'objet Vector de type IGraphicsData (Vector.<IGraphicsData>), qu'il est possible de réutiliser comme données source d'autres occurrences de formes ou pour stocker des informations de dessin pour un usage ultérieur.

Vous remarquerez qu'il existe plusieurs classes de remplissage (Fill) correspondant à chaque style de remplissage, mais une seule classe de trait (Stroke). ActionScript propose une seule classe de trait IGraphicsData car elle définit son style sur la base des classes de remplissage. Chaque trait allie donc la classe de trait à une classe de remplissage. Pour le reste, l'API de ces classes de données graphiques est identique aux méthodes qu'elles représentent dans la classe flash.display.Graphics :

Méthode graphique	**Classe de données**
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

En outre, la classe GraphicsPath possède ses propres méthodes d'utilitaire (`GraphicsPath.moveTo()`, `GraphicsPath.lineTo()`, `GraphicsPath.curveTo()`, `GraphicsPath.wideLineTo()` et `GraphicsPath.wideMoveTo()`) pour simplifier la définition de ces commandes pour une occurrence de GraphicsPath. Ces méthodes facilitent la définition ou la mise à jour directe des commandes et des valeurs de données.

Lorsque vous disposez d'une série d'occurrences d' IGraphicsData, utilisez la méthode `Graphics.drawGraphicsData()` pour rendre les graphiques. La méthode `Graphics.drawGraphicsData()` exécute séquentiellement un vecteur des occurrences d'IGraphicsData via l'API de dessin :

```
// 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(1,2,2);
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);
```

En modifiant une valeur du tracé utilisé par le dessin de l'exemple, il est possible de retracer la forme plusieurs fois pour obtenir une image plus complexe :

```
// 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);
```

Bien que les objets IGraphicsData puissent définir des styles de remplissage et de trait, ceux-ci ne sont pas obligatoires. Autrement dit, il est possible de définir des styles à l'aide des méthodes de la classe Graphics ou d'utiliser des objets IGraphicsData pour tracer un ensemble enregistré de tracés, et inversement.

Remarque : *la méthode* `Graphics.clear()` *permet d'effacer un dessin avant d'en commencer un nouveau, à moins que vous ne complétiez le dessin d'origine, comme l'illustre l'exemple ci-dessus. Lorsque vous modifiez une partie d'un tracé ou d'un ensemble d'objets IGraphicsData, retracez le dessin entier pour visualiser les changements.*

Lorsque vous utilisez des classes de données graphiques, le remplissage est rendu chaque fois que trois points au moins sont tracés car la forme est nécessairement fermée à ce point. Bien que le remplissage ait un effet de fermeture, ce n'est pas le cas du trait. Ce comportement est différent de celui de commandes `Graphics.lineTo()` ou `Graphics.moveTo()` utilisées plusieurs fois.

A propos de l'utilisation de drawTriangles()

Autre nouveauté de Flash Player 10 et d'Adobe AIR 1.5, la méthode avancée `Graphics.drawTriangles()` est similaire à la méthode `Graphics.drawPath()`. La méthode `Graphics.drawTriangles()` utilise également un objet Vector.<Number> pour spécifier les points permettant de dessiner un tracé.

La méthode `Graphics.drawTriangles()` a néanmoins pour but principal de faciliter la création d'effets tridimensionnels par le biais d'ActionScript. Pour plus d'informations sur l'utilisation de `Graphics.drawTriangles()` pour produire des effets tridimensionnels, consultez la section « Création d'effets 3D à l'aide de triangles » à la page 528.

Chapitre 15 : Utilisation de la géométrie

Le package flash.geom contient des classes qui définissent des objets géométriques, tels que des points, des rectangles et des matrices de transformation. Vous utilisez ces classes pour définir les propriétés des objets qui sont utilisés dans d'autres classes.

Principes de base de la géométrie

Introduction à l'utilisation de la géométrie

La géométrie peut être une matière difficile à appréhender et à retenir mais quelques connaissances peuvent faciliter l'utilisation d'ActionScript.

Le package flash.geom contient des classes qui définissent des objets géométriques, tels que des points, des rectangles et des matrices de transformation. Ces classes ne fournissent pas nécessairement de fonctionnalité par elles-mêmes ; néanmoins, elles sont utilisées pour définir les propriétés des objets utilisés dans d'autres classes.

Toutes les classes de géométrie se basent sur la notion selon laquelle les emplacements à l'écran sont représentés comme un plan en deux dimensions. L'écran est traité comme un graphique plat avec un axe horizontal (x) et un axe vertical (y). Tout emplacement (ou *point*) à l'écran peut être représenté sous la forme d'une paire de valeurs x et y, appelées *coordonnées* de cet emplacement.

Chaque objet d'affichage, y compris la scène, a son propre *espace de coordonnées*, essentiellement son propre graphique pour le tracé d'emplacements d'objets d'affichage enfant, de dessins, etc. Généralement, l'*origine* (coordonnées 0, 0 où les axes x et y se croisent) est placée en haut à gauche de l'objet d'affichage. Ceci est toujours vrai pour la scène, mais pas nécessairement pour d'autres objets d'affichage. Comme dans les systèmes de coordonnées en deux dimensions standard, les valeurs sur l'axe x augmentent en allant vers la droite et diminuent en allant vers la gauche - à gauche de l'origine, la coordonnée x est négative. Néanmoins, contrairement aux systèmes de coordonnées classiques, dans ActionScript, les valeurs sur l'axe y augmentent an allant vers le bas et diminuent en allant vers le haut de l'écran (avec des valeurs au-dessus de l'origine ayant une coordonnée y négative). Etant donné que l'angle supérieur gauche de la scène est l'origine de son espace de coordonnées, tout objet sur la scène aura une coordonnée x supérieure à 0 et inférieure à la largeur de la scène. Sa coordonnée y sera supérieure à 0 et inférieure à la hauteur de la scène.

Vous pouvez utiliser des occurrences de la classe Point pour représenter des points individuels dans un espace de coordonnées. Vous pouvez créer une occurrence de Rectangle pour représenter une région rectangulaire dans un espace de coordonnées. Les utilisateurs chevronnés peuvent utiliser une occurrence de Matrix pour appliquer des transformations multiples ou complexes à un objet d'affichage. De nombreuses transformations simples (rotation, position, et changements d'échelle, par exemple) peuvent être appliquées directement à un objet d'affichage à l'aide des propriétés de ce dernier. Pour plus d'informations sur l'application de transformations à l'aide des propriétés d'un objet d'affichage, consultez la section « Manipulation des objets d'affichage » à la page 299.

Tâches de géométrie courantes

Vous souhaiterez probablement effectuer les tâches suivantes à l'aide des classes de géométrie dans ActionScript :

- Calcul de la distance entre deux points
- Détermination des coordonnées d'un point dans différents espaces de coordonnées
- Déplacement d'un objet d'affichage à l'aide de l'angle et de la distance

- Utilisation des occurrences Rectangle :
  - Repositionnement d'une occurrence de Rectangle
  - Redimensionnement d'une occurrence de Rectangle
  - Détermination de la taille combinée ou des zones de chevauchement d'occurrences Rectangle
- Création d'objets Matrix
- Utilisation d'un objet Matrix pour appliquer des transformations à un objet d'affichage

Concepts importants et terminologie

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Coordonnées cartésiennes : les coordonnées sont généralement écrites sous la forme d'une paire de nombres (5, 12 ou 17, -23). Les deux nombres sont la coordonnée x et la coordonnée y, respectivement.
- Espace de coordonnées : représentation graphique des coordonnées contenues dans un objet d'affichage, par rapport auquel sont positionnés les éléments enfant.
- Origine : point dans un espace de coordonnées où l'axe x et l'axe y se croisent. Ce point a les coordonnées 0, 0.
- Point : emplacement unique dans un espace de coordonnées. Dans le système de coordonnées en 2 dimensions utilisé dans ActionScript, un point est défini par son emplacement le long de l'axe x et de l'axe y (les coordonnées du point).
- Point d'alignement : dans un objet d'affichage, l'origine (coordonnées 0, 0) de son espace de coordonnées.
- Echelle : taille relative d'un objet par rapport à sa taille d'origine. Mettre un objet à l'échelle consiste à modifier sa taille en l'étirant ou en le rétrécissant.
- Translation : modifier les coordonnées d'un point d'un espace de coordonnées à un autre.
- Transformation : modification des caractéristiques visuelles d'un graphique (rotation de l'objet, modification de son échelle, désalignement, déformation ou altération de sa couleur).
- Axe X : axe horizontal dans le système de coordonnées en 2 dimensions utilisé dans ActionScript.
- Axe Y : axe vertical dans le système de coordonnées en 2 dimensions utilisé dans ActionScript.

Utilisation des exemples fournis dans ce chapitre

Un grand nombre des exemples fournis dans ce chapitre illustrent des calculs ou des changements de valeurs ; la plupart d'entre eux incluent les appels de la fonction `trace()` appropriés pour démontrer les résultats du code. Pour tester ces exemples, procédez comme suit :

1. Créez un document vide à l'aide de l'outil de programmation Flash.
2. Sélectionnez une image-clé dans le scénario.
3. Ouvrez le panneau Actions et copiez le code dans le panneau Script.
4. Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

   Les résultats des fonctions `trace()` des codes s'affichent dans le panneau Sortie.

Certains exemples du chapitre démontrent l'application de transformations aux objets d'affichage. Les résultats de ces exemples sont affichés au lieu d'être entrés dans un champ de texte. Pour tester ces exemples de transformation, procédez comme suit :

1. Créez un document vide à l'aide de l'outil de programmation Flash.

2 Sélectionnez une image-clé dans le scénario.

3 Ouvrez le panneau Actions et copiez le code dans le panneau Script.

4 Créez une occurrence de symbole de clip sur la scène. Par exemple, dessinez une forme, sélectionnez-la et choisissez Modification > Convertir en symbole. Donnez ensuite un nom au symbole.

5 Sélectionnez le clip de la scène et donnez un nom à l'occurrence dans l'Inspecteur des Propriétés. Le nom doit correspondre au nom utilisé pour l'objet d'affichage dans l'exemple de code (par exemple, si le code applique une transformation à un objet `myDisplayObject`, vous devez appeler votre occurrence de clip `myDisplayObject` également).

6 Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

Les résultats des transformations appliquées à l'objet comme indiqué dans le code s'affichent à l'écran.

Ces techniques de test d'exemples de code sont décrites de manière plus détaillée dans « Test des exemples de code contenus dans un chapitre » à la page 36.

Utilisation des objets Point

Un objet Point définit une paire de coordonnées cartésiennes. Il représente un emplacement dans un système de coordonnées à deux dimensions, dans lequel *x* est l'axe horizontal et *y* l'axe vertical.

Pour définir un objet Point, vous définissez ses propriétés `x` et `y` comme suit :

```
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;
```

Calcul de la distance entre deux points

Vous pouvez utiliser la méthode `distance()` de la classe Point pour calculer la distance entre deux points dans un espace de coordonnées. Par exemple, le code suivant calcule la distance entre les points d'alignement de deux objets d'affichage, `circle1` et `circle2`, dans le même conteneur d'objet d'affichage :

```
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);
```

Translation d'espaces de coordonnées

Si deux objets d'affichage se trouvent dans des conteneurs d'objet d'affichage différents, il se peut qu'ils soient dans des espaces de coordonnées différents. Vous pouvez utiliser la méthode `localToGlobal()` de la classe DisplayObject pour traduire les coordonnées dans le même espace de coordonnées (global), celui de la scène. Par exemple, le code suivant calcule la distance entre les points d'alignement de deux objets d'affichage, `circle1` et `circle2`, dans les différents conteneurs d'objet d'affichage :

```
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);
```

De même, pour rechercher la distance du point d'alignement d'un objet d'affichage nommé `target` à partir d'un point spécifique sur la scène, vous pouvez utiliser la méthode `localToGlobal()` de la classe 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);
```

Déplacement d'un objet d'affichage d'une distance et d'un angle spécifiés

Vous pouvez utiliser la méthode `polar()` de la classe Point pour déplacer un objet d'affichage d'une distance et d'un angle spécifiques. Par exemple, le code suivant déplace l'objet `myDisplayObject` d'une distance de 100 pixels et d'un angle de 60 degrés :

```
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;
```

Autres utilisations de la classe Point

Vous pouvez utiliser des objets Point avec les propriétés et les méthodes suivantes :

Classe	Méthodes ou propriétés	Description
DisplayObjectContainer	`areInaccessibleObjectsUnderPoint()getObjectsUnderPoint()`	Utilisée pour renvoyer une liste d'objets sous un point dans un conteneur d'objet d'affichage.
BitmapData	`hitTest()`	Utilisée pour définir le pixel dans l'objet BitmapData ainsi que le point pour lequel vous recherchez une zone active.

Classe	Méthodes ou propriétés	Description

BitmapData	`applyFilter()`
	`copyChannel()`
	`merge()`
	`paletteMap()`
	`pixelDissolve()`
	`threshold()`
		Utilisée pour indiquer les positions des rectangles qui définissent les opérations.

Matrice	`deltaTransformPoint()` `transformPoint()`	Utilisée pour définir des points auxquels vous souhaitez appliquer une transformation.

Rectangle	`bottomRight`
	`size`
	`topLeft`
		Utilisée pour définir ces propriétés.

Utilisation des objets Rectangle

Un objet Rectangle définit une zone rectangulaire. Un objet Rectangle a une position définie par les coordonnées *x* et *y* de son angle supérieur gauche, une propriété `width` et une propriété `height`. Vous pouvez définir ces propriétés pour un nouvel objet Rectangle en appelant la fonction constructeur `Rectangle()`, comme suit :

```
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);
```

Redimensionnement et repositionnement des objets Rectangle

Il existe de nombreuses façons de redimensionner et de repositionner des objets Rectangle.

Vous pouvez redimensionner directement l'objet Rectangle en modifiant ses propriétés `x` et `y`. Ceci n'a aucune incidence sur la largeur ou la hauteur de l'objet Rectangle.

```
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)
```

Comme indiqué dans le code suivant, si vous modifiez la propriété `left` ou `top` d'un objet Rectangle, il est repositionné avec ses propriétés`x` et `y` correspondant aux propriétés `left` et `top`, respectivement. Néanmoins, la position de l'angle inférieur gauche de l'objet Rectangle ne change pas. Par conséquent, il est redimensionné.

```
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)
```

De même, comme indiqué dans l'exemple suivant, si vous modifiez la propriété `bottom` ou `right` d'un objet Rectangle, la position de son angle supérieur gauche ne change pas. Il est donc redimensionné en conséquence :

```
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)
```

Vous pouvez également repositionner un objet Rectangle à l'aide de la méthode `offset()`, comme suit :

```
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)
```

La méthode `offsetPt()` fonctionne de la même façon, sauf qu'elle prend un objet Point comme paramètre, plutôt que les valeurs de décalage *x* et *y*.

Vous pouvez également redimensionner un objet Rectangle à l'aide de la méthode `inflate()`, qui inclut deux paramètres, `dx` et `dy`. Le paramètre `dx` représente le nombre de pixels dont les côtés gauche et droit du rectangle vont se déplacer depuis le centre, et le paramètre `dy` représente le nombre de pixels dont les côtés supérieur et inférieur du rectangle vont se déplacer depuis le centre :

```
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)
```

La méthode `inflatePt()` fonctionne de la même façon, sauf qu'elle prend un objet Point comme paramètre, plutôt que les valeurs de décalage `dx` et `dy`.

Recherche d'unions et d'intersections d'objets Rectangle

Vous utilisez la méthode `union()` pour rechercher la région rectangulaire formée par les limites de deux rectangles :

```
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)
```

Vous utilisez la méthode `intersection()` pour rechercher la région rectangulaire formée par la région commune de deux rectangles :

```
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)
```

Vous utilisez la méthode `intersects()` pour savoir si deux rectangles se recouvrent. Vous pouvez également utiliser la méthode `intersects()` pour savoir si un objet d'affichage est dans une certaine région de la scène. Par exemple, dans le code suivant, supposez que l'espace de coordonnées du conteneur d'objet d'affichage contenant l'objet `circle` est identique à celui de la scène. L'exemple indique comment utiliser la méthode `intersects()` pour déterminer si un objet d'affichage, `circle`, recoupe des régions spécifiées de la scène, définies par les objets Rectangle `target1` et `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
```

De même, vous pouvez utiliser la méthode `intersects()` pour savoir si les cadres de délimitation de deux objets d'affichage se chevauchent. Vous pouvez utiliser la méthode `getRect()` de la classe DisplayObject pour inclure un espace supplémentaire que les traits d'un objet d'affichage peuvent ajouter à une région de délimitation.

Autres utilisations des objets Rectangle

Les objets Rectangle sont utilisés dans les propriétés et méthodes suivantes :

Classe	Méthodes ou propriétés	Description
BitmapData	`applyFilter()`, `colorTransform()`, `copyChannel()`, `copyPixels()`, `draw()`, `fillRect()`, `generateFilterRect()`, `getColorBoundsRect()`, `getPixels()`, `merge()`, `paletteMap()`, `pixelDissolve()`, `setPixels()` et `threshold()`	Utilisée comme type de certains paramètres pour définir une région de l'objet BitmapData.
DisplayObject	`getBounds()`, `getRect()`, `scrollRect`, `scale9Grid`	Utilisée comme type de données pour la propriété ou le type de données renvoyé.
PrintJob	`addPage()`	Utilisée pour définir le paramètre `printArea`.
Sprite	`startDrag()`	Utilisée pour définir le paramètre `bounds`.
TextField	`getCharBoundaries()`	Utilisée comme type de valeur renvoyé.
Transform	`pixelBounds`	Utilisée comme type de données.

Utilisation des objets Matrix

La classe Matrix représente une matrice de transformation qui détermine le mappage des points d'un espace de coordonnées à l'autre. Pour appliquer diverses transformations graphiques à un objet d'affichage, vous pouvez définir les propriétés d'un objet Matrix, puis appliquer cet objet à la propriété `matrix` d'un objet Transform que vous appliquez ensuite comme propriété `transform` de l'objet d'affichage. Ces fonctions de transformation incluent la translation (repositionnement de *x* et *y*), la rotation, le redimensionnement et l'inclinaison.

Définition des objets Matrix

Même si vous pouvez définir une matrice en ajustant directement les propriétés (`a`, `b`, `c`, `d`, `tx`, `ty`) d'un objet Matrix, il est plus facile d'utiliser la méthode `createBox()`. Cette méthode comporte des paramètres qui vous permettent de définir directement les effets de redimensionnement, de rotation et de translation de la matrice résultante. Par exemple, le code suivant crée un objet Matrix qui a l'effet de redimensionner un objet horizontalement de 2,0, verticalement de 3,0, de le faire pivoter de 45 degrés, de le déplacer (translation) de 10 pixels vers la droite et de 20 pixels vers le bas :

```
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);
```

Vous pouvez également ajuster les effets de redimensionnement, de rotation et de translation d'un objet Matrix à l'aide des méthodes `scale()`, `rotate()` et `translate()`. Ces méthodes sont combinées aux valeurs de l'objet Matrix existant. Par exemple, le code suivant définit un objet Matrix qui redimensionne un objet d'un facteur de 4 et le fait pivoter de 60 degrés, car les méthodes `scale()` et `rotate()` sont appelées deux fois :

```
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;
```

Pour appliquer une transformation par inclinaison à un objet Matrix, ajustez sa propriété `b` ou `c`. Lorsque vous ajustez la propriété `b`, la matrice est inclinée verticalement et lorsque vous ajustez la propriété `c`, elle est inclinée horizontalement. Le code suivant incline l'objet Matrix `myMatrix` verticalement d'un facteur de 2 :

```
var skewMatrix:Matrix = new Matrix();
skewMatrix.b = Math.tan(2);
myMatrix.concat(skewMatrix);
```

Vous pouvez appliquer une transformation Matrix à la propriété `transform` d'un objet d'affichage. Par exemple, le code suivant applique une transformation Matrix à un objet d'affichage nommé `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;
```

La première ligne définit un objet Matrix sur la matrice de transformation existante utilisée par l'objet d'affichage `myDisplayObject` (la propriété `matrix` de la propriété `transformation` de l'objet d'affichage `myDisplayObject`). De cette façon, les méthodes de la classe Matrix que vous appelez ont un effet cumulatif sur la position, le redimensionnement et la rotation de l'objet d'affichage.

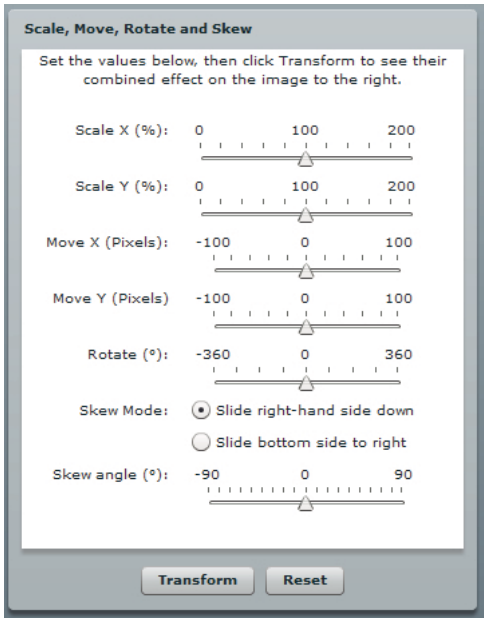
Remarque : *la classe ColorTransform est également comprise dans le package flash.geometry. Cette classe sert à définir la propriété `colorTransform` d'un objet Transform. Etant donné qu'elle n'applique aucun type de transformation géométrique, elle n'est pas traitée dans ce chapitre. Pour plus d'informations, consultez la classe ColorTransform dans le Guide de référence du langage et des composants ActionScript 3.0.*

Exemple : application d'une transformation de matrice à un objet d'affichage

L'exemple d'application DisplayObjectTransformer présente de nombreuses fonctions permettant d'utiliser la classe Matrix pour transformer un objet d'affichage, notamment :

- Rotation de l'objet d'affichage
- Redimensionnement de l'objet d'affichage
- Translation (repositionnement) de l'objet d'affichage
- Inclinaison de l'objet d'affichage

L'application fournit une interface permettant d'ajuster les paramètres de la transformation de matrice, comme suit :



Lorsque l'utilisateur clique sur le bouton de transformation, l'application applique la transformation appropriée.

L'objet d'affichage original et l'objet d'affichage pivoté de -45° et redimensionné de 50 %

Pour obtenir les fichiers d'application associés à cet exemple, voir www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application DisplayObjectTransformer se trouvent dans le dossier Samples/DisplayObjectTransformer. L'application se compose des fichiers suivants :

Fichier	Description
DisplayObjectTransformer.mxml ou DisplayObjectTransformer.fla	Fichier d'application principal en FLA pour Flash ou en MXML pour Flex
com/example/programmingas3/geometry/MatrixTransformer.as	Une classe qui contient des méthodes permettant d'appliquer des transformations de matrice.
img/	Un répertoire contenant des exemples de fichiers image utilisés par l'application.

Définition de la classe MatrixTransformer

La classe MatrixTransformer comprend des méthodes statiques qui appliquent des transformations géométriques d'objets Matrix.

Méthode transform()

La méthode `transform()` comprend des paramètres associés à chaque élément suivant :

- `sourceMatrix`—La matrice d'entrée, que la méthode transforme
- `xScale` et `yScale`—Le facteur de redimensionnement *x* et *y*
- `dx` et `dy`—Les montants de translation *x* et *y*, en pixels
- `rotation`—Le montant de rotation, en degrés
- `skew`—Le facteur d'inclinaison, en pourcentage
- `skewType`—Le sens d'inclinaison, `"right"` ou `"left"`

La valeur renvoyée est la matrice résultante.

La méthode `transform()` appelle les méthodes statiques suivantes de la classe :

- `skew()`
- `scale()`
- `translate()`
- `rotate()`

Chacune renvoie la matrice source avec la transformation appliquée.

Méthode skew()

La méthode `skew()` incline la matrice en ajustant les propriétés `b` et `c` de la matrice. Un paramètre facultatif, `unit`, détermine les unités utilisées pour définir l'angle d'inclinaison et, le cas échéant, la méthode convertit la valeur `angle` en radians :

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
```

Un objet Matrix `skewMatrix` est créé et ajusté pour appliquer la transformation par inclinaison. Au départ, il s'agit de la matrice d'identité, comme suit :

```
var skewMatrix:Matrix = new Matrix();
```

Le paramètre `skewSide` détermine le côté auquel l'inclinaison est appliquée. S'il est défini sur `"right"`, le code suivant définit la propriété `b` de la matrice :

```
skewMatrix.b = Math.tan(angle);
```

Autrement, le côté inférieur est incliné en ajustant la propriété `c` de la matrice, comme suit :

```
skewMatrix.c = Math.tan(angle);
```

L'inclinaison résultante est ensuite appliquée à la matrice existante en concaténant les deux matrices, comme indiqué dans l'exemple suivant :

```
sourceMatrix.concat(skewMatrix);
return sourceMatrix;
```

Méthode scale()

Comme le montre l'exemple suivant, la méthode `scale()` ajuste d'abord le facteur de redimensionnement s'il est fourni sous la forme d'un pourcentage, puis utilise la méthode `scale()` de l'objet matrix :

```
if (percent)
{
    xScale = xScale / 100;
    yScale = yScale / 100;
}
sourceMatrix.scale(xScale, yScale);
return sourceMatrix;
```

Méthode translate()

La méthode `translate()` applique simplement les facteurs de translation `dx` et `dy` en appelant la méthode `translate()` de l'objet matrix, comme suit :

```
sourceMatrix.translate(dx, dy);
return sourceMatrix;
```

Méthode rotate()

La méthode `rotate()` convertit le facteur de rotation d'entrée en radians (s'il est fourni en degrés ou dégradés), puis appelle la méthode `rotate()` de l'objet matrix :

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
sourceMatrix.rotate(angle);
return sourceMatrix;
```

Appel de la méthode MatrixTransformer.transform() depuis l'application

L'application contient une interface utilisateur permettant d'obtenir les paramètres de transformation de l'utilisateur. Elle les transmet ensuite, avec la propriété `matrix` de la propriété `transform` de l'objet d'affichage, à la méthode `Matrix.transform()`, comme suit :

```
tempMatrix = MatrixTransformer.transform(tempMatrix,
    xScaleSlider.value,
    yScaleSlider.value,
    dxSlider.value,
    dySlider.value,
    rotationSlider.value,
    skewSlider.value,
    skewSide );
```

L'application applique ensuite la valeur renvoyée à la propriété `matrix` de la propriété `transform` de l'objet d'affichage. Ainsi, la transformation est déclenchée :

```
img.content.transform.matrix = tempMatrix;
```

Chapitre 16 : Filtrage des objets d'affichage

Historiquement, l'application d'effets de filtres à des images bitmap est du domaine des logiciels spécialisés en retouche d'image, comme Adobe Photoshop® et Adobe Fireworks®. ActionScript 3.0 comporte le package flash.filters, qui contient une série de classes de filtres à effet pour les images bitmap, permettant ainsi aux développeurs d'appliquer par programmation des filtres aux bitmaps et aux objets d'affichage afin d'obtenir une grande partie des effets disponibles dans les applications de retouche graphique.

Principes de base du filtrage des objets d'affichage

Introduction au filtrage des objets d'affichage

L'une des façons de rendre une application plus séduisante est de lui ajouter des effets graphiques simples, comme une ombre portée derrière une photo pour créer l'illusion de la 3D, ou un rayonnement autour d'un bouton pour indiquer que c'est le bouton actif. ActionScript 3.0 comporte neuf filtres qui peuvent être appliqués à n'importe quel objet d'affichage ou à une occurrence de BitmapData. Ces filtres vont des effets de base (filtres Ombre portée et Rayonnement) à des effets plus complexes permettant de créer divers effets, tels que le nouveau filtre Mappage du déplacement et le filtre Convolution matricielle.

Tâches de filtrage courantes

Les opérations suivantes, qui sont décrites dans ce chapitre, peuvent être aisément accomplies en ActionScript à l'aide de filtres :

- Création d'un filtre
- Application d'un filtre à un objet d'affichage
- Suppression d'un filtre appliqué à un objet d'affichage
- Application d'un filtre aux données d'image d'une occurrence de BitmapData
- Suppression des filtres appliqués à un objet
- Création d'effets divers, tels que le rayonnement, le flou, l'ombre portée, la netteté, le déplacement, la détection de contour, l'estampage et autres effets

Concepts importants et terminologie

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Biseau : effet de rebord créé en éclaircissant les pixels de deux côtés contigus et en assombrissant les pixels des deux côtés opposés, afin de créer un effet tridimensionnel de bordure fréquemment utilisé pour donner à des boutons et autres graphiques un effet enfoncé ou sorti.
- Convolution : distorsion des pixels d'une image obtenue en combinant la valeur de chaque pixel avec celle(s) d'un ou plusieurs des pixels voisins, selon divers pourcentages.
- Déplacement : décalage des pixels d'une image à une nouvelle position.
- Matrice : grille de chiffres utilisée pour effectuer certains calculs mathématiques, en appliquant ces chiffres à diverses valeurs et en combinant le résultat.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester les exemples de code fournis. Etant donné que ce chapitre décrit comment créer et manipuler du contenu visuel, le test du code implique l'exécution du code et l'affichage des résultats dans le fichier SWF créé. La quasi-totalité des exemples créent du contenu par le biais de l'API de dessin ou chargent des images auxquelles sont appliqués des filtres.

Pour tester les codes de ce chapitre :

1. Créez un document vide à l'aide de l'outil de programmation Flash.
2. Sélectionnez une image-clé dans le scénario.
3. Ouvrez le panneau Actions et copiez le code dans le panneau Script.
4. Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

   Les résultats du code apparaissent dans le fichier SWF créé.

La quasi-totalité des exemples contenant du code qui crée une image bitmap, vous pouvez tester directement le code sans avoir à fournir un contenu bitmap. Vous pouvez également modifier le code pour charger vos propres images, afin de substituer ces dernières aux images utilisées dans les exemples.

Création et application de filtres

Les filtres permettent d'appliquer divers effets (ombre portée, biseau, flou, etc.) à des images bitmap et à des objets d'affichage. Chaque filtre étant défini sous forme de classe, il suffit pour appliquer un filtre de créer une occurrence d'un objet filtre, ce qui n'est guère différent de la création de tout autre objet. Après avoir créé une occurrence d'un objet filtre, il est facile de l'appliquer à un objet d'affichage à l'aide de la propriété `filters` de cet objet ou, dans le cas d'un objet BitmapData, de sa méthode `applyFilter()`.

Création d'un filtre

Pour créer un objet filtre, il suffit d'appeler la fonction constructeur de la classe du filtre voulu. Par exemple, pour créer un objet DropShadowFilter, utilisez le code suivant :

```
import flash.filters.DropShadowFilter;
var myFilter:DropShadowFilter = new DropShadowFilter();
```

Bien que cela n'apparaisse pas dans cet exemple, le constructeur de `DropShadowFilter()` (comme tous les autres constructeurs des classes de filtres) accepte plusieurs paramètres facultatifs qui permettent de modifier l'aspect de l'effet du filtre.

Application d'un filtre

Lorsque l'objet filtre a été créé, vous pouvez l'appliquer à un objet d'affichage ou à un objet BitmapData, mais le mode d'application du filtre dépend de l'objet concerné.

Application d'un filtre à un objet d'affichage

Pour appliquer un effet de filtrage à un objet d'affichage, utilisez sa propriété `filters`. La propriété `filters` d'un objet d'affichage est une occurrence de l'objet Array, dont les éléments sont les objets filtres appliqués à l'objet d'affichage. Pour appliquer un seul filtre à un objet d'affichage, créez l'occurrence de ce filtre, ajoutez-la à une occurrence d'Array, et affectez cet objet Array à la propriété `filters` de l'objet d'affichage :

```
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;
```

Pour affecter plusieurs filtres à l'objet, il suffit d'ajouter tous ces filtres à l'occurrence d'Array avant de l'affecter à la propriété `filters`. Vous pouvez ajouter plusieurs objets à un objet Array en les passant en paramètres à son constructeur. Par exemple, ce code applique un filtre Biseau et un filtre Rayonnement à l'objet d'affichage créé précédemment :

```
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;
```

Pour créer le tableau des filtres, vous pouvez utiliser le constructeur `new Array()` (comme dans les exemples précédents) ou la syntaxe littérale Array, en mettant les filtres entre crochets (`[]`). Par exemple, cette ligne de code :

```
var filters:Array = new Array(dropShadow, blur);
```

donne un résultat identique à celle-ci :

```
var filters:Array = [dropShadow, blur];
```

Si vous appliquez plusieurs filtres à des objets d'affichage, ils sont appliqués en séquence, de manière cumulative. Par exemple, si un tableau de filtres comporte deux éléments, le filtre Biseau puis le filtre Ombre portée, ce dernier est appliqué à la fois au filtre Biseau et à l'objet d'affichage lui-même, du fait que le filtre Ombre portée est en seconde position dans le tableau des filtres. Si vous souhaitez appliquer des filtres de manière non cumulative, appliquez chaque filtre à une nouvelle copie de l'objet d'affichage.

Pour affecter uniquement un ou quelques filtres à un objet d'affichage, vous pouvez créer l'occurrence du filtre et l'affecter à l'objet dans la même instruction. Par exemple, le code suivant applique un filtre Flou à un objet d'affichage nommé `myDisplayObject` :

```
myDisplayObject.filters = [new BlurFilter()];
```

Le code précédent crée une occurrence d'Array en utilisant la syntaxe littérale d'Array (entre crochets), puis crée une nouvelle occurrence de BlurFilter comme élément du tableau, et affecte ce dernier à la propriété `filters` de l'objet d'affichage `myDisplayObject`.

Suppression des filtres appliqués à un objet

Pour supprimer tous les filtres d'un objet d'affichage, il suffit d'affecter la valeur null à la propriété `filters` de celui-ci :

```
myDisplayObject.filters = null;
```

Si vous avez appliqué plusieurs filtres à un objet et souhaitez n'en supprimer qu'un, plusieurs étapes sont nécessaires pour modifier le tableau de la propriété `filters`. Pour plus d'informations, consultez la section « Problèmes potentiels d'utilisation des filtres » à la page 366.

Application d'un filtre à un objet BitmapData

L'application d'un filtre à un objet BitmapData nécessite d'utiliser la méthode `applyFilter()` de l'objet BitmapData :

```
var rect:Rectangle = new Rectangle();
var origin:Point = new Point();
myBitmapData.applyFilter(sourceBitmapData, rect, origin, new BlurFilter());
```

La méthode `applyFilter()` applique un filtre à un objet source BitmapData, produisant ainsi une nouvelle image filtrée. Cette méthode ne modifie pas l'image originale, car le résultat de l'application du filtre à celle-ci est enregistré dans l'occurrence de BitmapData pour laquelle la méthode `applyFilter()` est appelée.

Fonctionnement des filtres

Le filtrage des objets d'affichage consiste à mettre en cache une copie de l'objet original sous forme d'un bitmap transparent.

Lorsqu'un filtre a été appliqué à un objet d'affichage, Adobe Flash Player ou Adobe® AIR™ conserve en cache l'objet sous forme de bitmap tant que cet objet possède une liste de filtres valide. Le bitmap source est ensuite repris en tant qu'image originale pour les effets de filtrage suivants.

Tout objet d'affichage comporte généralement deux bitmaps : le premier avec l'objet d'affichage source non filtré d'origine et un autre pour l'image finale après filtrage. L'image finale est utilisée pour le rendu. Tant que l'objet d'affichage ne change pas, l'image source ne nécessite aucune actualisation.

Problèmes potentiels d'utilisation des filtres

Plusieurs sources potentielles de confusion ou de problèmes peuvent survenir lors de l'utilisation de filtres. Elles sont décrites dans les sections suivantes.

Filtres et mise en cache bitmap

Pour appliquer un filtre à un objet d'affichage, vous devez activer la mise en cache sous forme de bitmap pour cet objet. Si vous appliquez un filtre à un objet d'affichage dont la propriété `cacheAsBitmap` est `false`, Flash Player ou AIR définit automatiquement cette propriété `cacheAsBitmap` sur `true`. Si vous supprimez ensuite tous les filtres de l'objet, Flash Player ou AIR rend à la propriété `cacheAsBitmap` sa valeur précédente.

Modification des filtres à l'exécution

Si un ou plusieurs filtres sont déjà appliqués à un objet d'affichage, vous ne pouvez pas modifier le jeu de filtres en ajoutant d'autres filtres ou en supprimant des filtres existants du tableau de la propriété `filters`. Pour modifier le jeu de filtres appliqué ou lui ajouter des filtres, vous devez plutôt effectuer les modifications requises dans un tableau distinct, puis l'assigner à la propriété filters de l'objet d'affichage pour que les filtres soient appliqués à l'objet. La procédure la plus simple consiste à lire le tableau de la propriété `filters` dans une variable Array, puis à effectuer les modifications requises dans ce tableau temporaire. Vous réassignez alors ce tableau à la propriété `filters` de l'objet d'affichage. Dans les cas de figure plus complexes, il peut s'avérer nécessaire de conserver un tableau maître distinct de filtres. Vous effectuez toute modification requise dans ce tableau maître de filtres, puis vous le réassignez à la propriété `filters` de l'objet d'affichage après chaque modification.

Ajout d'un autre filtre

Le code suivant illustre le processus d'ajout d'un autre filtre à un objet d'affichage auquel sont déjà appliqués un ou plusieurs filtres. Initialement, un filtre Rayonnement est appliqué à l'objet d'affichage `myDisplayObject`. Lorsque l'utilisateur clique sur l'objet d'affichage, la fonction `addFilters()` est appelée. Dans cette fonction, deux filtres supplémentaires sont appliqués à `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);
```

Suppression d'un filtre dans un jeu de filtres

Si plusieurs filtres sont appliqués à un objet d'affichage et que vous souhaitez supprimer l'un des filtres sans affecter les autres filtres appliqués, vous copiez les filtres dans un tableau temporaire, vous supprimez le filtre superflu du tableau, puis vous réassignez le tableau temporaire à la propriété `filters` de l'objet d'affichage. La section « Récupération des valeurs et suppression des éléments du tableau » à la page 165 décrit plusieurs techniques de suppression d'un ou de plusieurs éléments de tout tableau.

La technique la plus simple consiste à supprimer le filtre en tête de pile appliqué à l'objet (en d'autres termes, le dernier filtre appliqué à ce dernier). Vous utilisez la méthode `pop()` de la classe Array pour supprimer le filtre du tableau :

```
// 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;
```

Pour supprimer le filtre en bas de pile (en d'autres termes, le premier filtre appliqué à l'objet), vous utilisez le même code en substituant la méthode `shift()` de la classe Array à la méthode `pop()`.

Pour supprimer un filtre figurant au centre d'un tableau de filtres (sous réserve que le tableau contienne plus de deux filtres), vous disposez de la méthode `splice()`. Vous devez connaître l'index (la position dans le tableau) du filtre à supprimer. Par exemple, le code suivant supprime le deuxième filtre (doté de l'index 1) d'un objet d'affichage :

```
// 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;
```

Identification de l'index d'un filtre

Vous devez connaître l'index du filtre à supprimer du tableau. A cet effet, il est nécessaire d'identifier (selon le mode de conception de l'application) ou de calculer l'index du filtre à supprimer.

L'approche recommandée consiste à concevoir votre application de sorte que le filtre à supprimer occupe systématiquement la même position dans le jeu de filtres. Par exemple, si vous disposez d'un objet d'affichage unique auquel sont appliqués un filtre Convolution et un filtre Ombre portée (dans cet ordre) et que vous souhaitez supprimer le filtre Ombre portée tout en conservant le filtre Convolution, vous connaissez la position occupée par le filtre (première). Vous savez donc à l'avance quelle méthode Array utiliser (soit, dans ce cas, `Array.pop()` pour supprimer le filtre Ombre portée).

Si le filtre à supprimer est toujours d'un type déterminé, mais qu'il n'occupe pas systématiquement la même position dans le jeu de filtres, vous pouvez vérifier le type de données de chaque filtre du tableau pour identifier le filtre à supprimer. Par exemple, le code suivant identifie le filtre Rayonnement dans un jeu de filtres et le supprime de ce dernier.

```
// 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;
```

Dans un cas de figure plus complexe (si le filtre à supprimer est sélectionné à l'exécution, par exemple), l'approche recommandée consiste à conserver une copie distincte et permanente du tableau de filtres, qui sert de liste maîtresse de filtres. Lorsque vous modifiez le jeu de filtres (ajout ou suppression de filtre), modifiez la liste maîtresse, puis appliquez ce tableau de filtres en tant que propriété `filters` de l'objet d'affichage.

Par exemple, dans le code ci-après, plusieurs filtres Convolution sont appliqués à un objet d'affichage pour créer divers effets visuels et l'un de ces filtres est supprimé ultérieurement dans l'application sans affecter les autres filtres. Dans ce cas de figure, le code conserve une copie maîtresse du tableau de filtres, ainsi qu'une référence au filtre à supprimer. Rechercher et identifier le filtre approprié est similaire à l'approche précédente, excepté qu'au lieu d'effectuer une copie temporaire du tableau de filtres, la copie maîtresse est manipulée, puis appliquée à l'objet d'affichage.

```
// 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;
}
```

Si vous adoptez cette approche (qui consiste à comparer une référence stockée à un filtre aux éléments que contient le tableau de filtres pour identifier le filtre à supprimer), vous *devez* conserver une copie distincte du tableau de filtres. En effet, le code ne fonctionne pas si vous comparez la référence stockée au filtre aux éléments d'un tableau temporaire copié dans la propriété `filters` de l'objet d'affichage. La raison en est simple : lorsque vous assignez un tableau à la propriété `filters`, Flash Player ou AIR effectue en interne une copie de chaque objet filtre du tableau. Ces copies (plutôt que les objets d'origine) sont appliquées à l'objet d'affichage. Lorsque vous lisez la propriété `filters` dans un tableau temporaire, celui-ci contient des références aux objets filtre copiés plutôt qu'aux objets filtre d'origine. Par conséquent, si vous tentez dans l'exemple précédent de déterminer l'index de `filterToRemove` en le comparant aux filtres d'un tableau de filtres temporaire, aucune correspondance n'est détectée.

Filtres et transformations d'objets

Les zones filtrées (ombres portées, par exemple) situées hors du cadre de sélection d'un objet d'affichage ne sont pas prises en considération pour la détection de clics (chevauchement ou intersection de deux occurrences). La méthode de détection des clics de la classe DisplayObject étant de type vectoriel, il est impossible d'en pratiquer une sur le bitmap résultant. Par exemple, si vous appliquez un filtre Biseau à une occurrence de bouton, la détection des clics n'est pas possible sur la partie biseautée de l'occurrence.

Le redimensionnement, la rotation et l'inclinaison ne sont pas pris en charge par les filtres ; si l'objet d'affichage filtré lui-même est redimensionné (`scaleX` et `scaleY` différents de 100 %), l'effet de filtre n'est pas redimensionné avec l'occurrence. La forme originale de l'occurrence est certes pivotée, inclinée ou redimensionnée, mais pas le filtre.

Vous pouvez animer une occurrence avec un filtre afin de créer des effets réalistes, ou imbriquer des occurrences et utiliser la classe BitmapData pour animer des filtres afin d'obtenir ces effets.

Filtres et objets bitmaps

Si vous appliquez un filtre à un objet BitmapData, la propriété `cacheAsBitmap` de cet objet est automatiquement `true`. Le filtre peut ainsi être appliqué à la copie de l'objet plutôt qu'à ce dernier.

Cette copie est alors placée à l'écran par-dessus l'objet original, aussi près que possible, au pixel près. Si les limites du bitmap original changent, la copie à laquelle le filtrage est appliqué est recréée à partir de l'original au lieu d'être étirée.

Si vous supprimez tous les filtres de l'objet d'affichage, la propriété `cacheAsBitmap` retrouve sa valeur d'origine.

Filtres d'affichage disponibles

ActionScript 3.0 comprend dix classes de filtre que vous pouvez appliquer aux objets d'affichage et aux objets BitmapData :

- filtre Biseau (classe BevelFilter)
- filtre Flou (classe BlurFilter)
- filtre Ombre portée (classe DropShadowFilter)
- filtre Rayonnement (classe GlowFilter)
- filtre Biseau dégradé (classe GradientBevelFilter)
- filtre Rayonnement dégradé (classe GradientGlowFilter)
- filtre Matrice de couleurs (classe ColorMatrixFilter)
- filtre Convolution (classe ConvolutionFilter)
- filtre Mappage de déplacement (classe DisplacementMapFilter)
- filtre Shader (classe ShaderFilter)

Les six premiers sont des filtres simples pouvant être utilisés pour des effets spécifiques, avec certains réglages possibles. Ces six filtres peuvent être appliqués en ActionScript, mais aussi à partir du panneau Filtres d'Adobe Flash CS4 Professional. Par conséquent, même si vous appliquez des filtres à l'aide d'ActionScript, sous réserve de disposer de l'outil de programmation Flash, vous pouvez utiliser son interface visuelle pour vérifier rapidement l'effet des différents filtres et de leurs réglages afin de créer l'effet désiré.

Les quatre derniers filtres sont uniquement disponibles en ActionScript. Ces filtres (filtre Matrice de couleurs, filtre Convolution, filtre Mappage de déplacement et filtre Shader) permettent de créer des types d'effet beaucoup plus souples. Plutôt que proposer un effet unique optimisé, ils assurent puissance et souplesse. Par exemple, en choisissant différentes valeurs pour sa matrice, il est possible d'utiliser le filtre Convolution pour créer des effets de flou, de gravure, d'accentuation, mais aussi pour des transformations, la détection de contour des couleurs, etc.

Chacun de ces filtres, qu'il soit simple ou complexe, dispose de propriétés dont les valeurs peuvent être modifiées. En général, il existe deux possibilités pour définir les propriétés d'un filtre. Tous les filtres permettent de définir leurs propriétés en passant des valeurs en paramètres au constructeur de l'objet filtre. Que les propriétés du filtre soient définies ou non par un passage de paramètres, il est aussi possible de modifier ultérieurement ses réglages en changeant les valeurs des propriétés de l'objet filtre. La plupart des exemples de code définissent directement les propriétés, afin de faciliter la compréhension de l'exemple. Néanmoins, il est également possible, en général, d'obtenir le même résultat avec moins de lignes de code, en passant les valeurs en paramètres dans le constructeur de l'objet filtre. Pour plus d'informations sur les caractéristiques de chaque filtre, ses propriétés et les paramètres constructeurs correspondants, consultez les codes associés au package flash.filters dans le Guide de référence du langage et des composants ActionScript 3.0.

Filtre Biseau

La classe BevelFilter vous permet d'ajouter une bordure en relief à l'objet filtré. Avec ce filtre, les angles et les côtés des objets semblent ciselés, biseautés.

Les propriétés de la classe BevelFilter permettent de modifier l'apparence du biseau. Vous pouvez définir les couleurs des zones claires et sombres, l'adoucissement et les angles des côtés du biseau, ainsi que la tille de ces derniers. Vous pouvez même créer un effet de poinçonnage.

L'exemple suivant charge une image externe et lui applique un filtre Biseau.

```
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];
```

Filtre Flou

La classe BlurFilter ajoute un effet de flou à un objet d'affichage et son contenu. Les effets de flou permettent de donner l'impression qu'un objet n'est pas dans le plan de mise au point ou de simuler l'effet d'un mouvement rapide (flou de mouvement). En choisissant une valeur faible pour la propriété `quality`, vous pouvez simuler un effet de photo légèrement floue. Le choix d'une valeur élevée pour la propriété `quality` permet d'obtenir un effet de flou léger proche de celui d'un flou gaussien.

L'exemple suivant crée un objet cercle à l'aide de la méthode `drawCircle()` de la classe Graphics, puis lui applique un filtre Flou :

```
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];
```

Filtre Ombre portée

L'ombre portée donne l'impression d'une source lumineuse ponctuelle au-dessus de l'objet cible. Il est possible de modifier la position et l'intensité de cette source lumineuse pour produire divers effets d'ombre portée.

La classe DropShadowFilter utilise un algorithme similaire à celui du filtre Flou. La principale différence tient au fait que le filtre Ombre portée possède quelques propriétés supplémentaires qui permettent de simuler diverses caractéristiques d'une source lumineuse (canal alpha, couleur, décalage et luminosité).

Le filtre Ombre portée permet aussi d'appliquer des options de transformation au style de l'ombre portée (ombre interne ou externe, ou mode de masquage).

Le code suivant crée un objet Sprite carré et lui applique un filtre Ombre portée.

```
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];
```

Filtre Rayonnement

La classe GlowFilter applique un effet d'éclairage aux objets d'affichage afin de suggérer qu'une lumière est braquée à partir du dessous de l'objet pour créer un faible rayonnement.

Comme le filtre Ombre portée, le filtre Rayonnement possède des propriétés qui permettent de modifier la distance, l'angle et la couleur de la source lumineuse en fonction de l'effet désiré. L'objet GlowFilter comporte aussi plusieurs options pour modifier le style de rayonnement, notamment le rayonnement interne ou externe et le mode de masquage.

Le code suivant crée un objet Sprite en forme de croix et lui applique un filtre Rayonnement.

```
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];
```

Filtre Biseau dégradé

La classe GradientBevelFilter vous permet d'appliquer un effet de biseau optimisé aux objets d'affichage ou aux objets BitmapData. L'utilisation d'un dégradé de couleurs sur le biseau améliore beaucoup l'effet de relief de celui-ci, en donnant aux côtés un aspect 3D plus réaliste.

L'exemple suivant crée un objet rectangle à l'aide de la méthode `drawRect()` de la classe Shape, puis lui applique un filtre Biseau dégradé :

```
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);
```

Filtre Rayonnement dégradé

La classe GradientGlowFilter vous permet d'appliquer un effet de rayonnement optimisé aux objets d'affichage ou aux objets BitmapData. Cet effet donne davantage de contrôle sur le rayonnement, produisant ainsi un effet plus réaliste. De plus, le filtre Rayonnement dégradé permet d'appliquer un rayonnement aux côtés intérieur, extérieur ou supérieur de l'objet.

L'exemple suivant dessine un cercle auquel un filtre Rayonnement dégradé est appliqué. Le montant de flou horizontal et vertical augmente à mesure que le pointeur de la souris s'approche du coin inférieur droit de la scène. Si l'utilisateur clique dans la scène, le niveau de flou augmente.

```
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);
```

Exemple : combinaison de filtres de base

L'exemple de code suivant applique plusieurs filtres de base, en combinaison avec un timer pour la création d'actions répétitives, pour obtenir la simulation d'un feu de circulation.

```
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();
```

Filtre Matrice de couleurs

La classe ColorMatrixFilter permet de manipuler les valeurs de couleur et les valeurs alpha des objets filtrés. Il est ainsi possible de créer des changements de saturation, des rotations de teinte (passage d'une palette d'une plage de couleur à une autre), de définir la luminance de la couche alpha et de produire d'autres effets de manipulation des couleurs en utilisant les valeurs d'un canal couleur pour les appliquer aux autres canaux.

Le principe de fonctionnement de ce filtre est le suivant : les pixels de l'image source sont analysés un par un et leurs composants rouge, vert, bleu et alpha sont séparés. Les valeurs de la matrice de couleur sont alors multipliées par chacune de ces valeurs, et les résultats sont ajoutés pour déterminer la valeur colorimétrique résultante qui sera affichée à l'écran pour ce pixel. La propriété `matrix` du filtre est un tableau de 20 nombres qui sont utilisés pour le calcul de la couleur finale. Pour plus d'informations sur l'algorithme utilisé pour calculer les valeurs de couleur, consultez la description de la propriété matrix de la classe ColorMatrixFilter dans le Guide de référence du langage et des composants ActionScript 3.0.

Vous trouverez plus d'informations et des exemples du filtre Matrice de couleurs dans l'article « Using Matrices for Transformations, Color Adjustments, and Convolution Effects in Flash » sur le site Web du Centre de développement d'Adobe.

Filtre Convolution

La classe ConvolutionFilter permet d'appliquer un large éventail de transformations de traitement d'images aux objets BitmapData ou aux objets d'affichage, tels que la définition du flou, la détection du contour, l'accentuation, l'estampage et le biseautage.

Le principe de fonctionnement du filtre Convolution est le suivant : les pixels de l'image source sont analysés un par un pour déterminer la couleur finale de ce pixel sur la base de sa valeur et de celles des pixels adjacents. Une matrice, sous forme d'un tableau de valeurs numériques, indique dans quelle mesure la valeur de chaque pixel adjacent particulier affecte la valeur finale.

Le type de matrice le plus fréquemment utilisé est un tableau de trois par trois. La matrice comporte donc neuf valeurs :

```
N   N   N
N   P   N
N   N   N
```

Lorsque Flash Player ou AIR applique le filtre Convolution à un pixel donné, il analyse la valeur colorimétrique du pixel lui-même (« P » dans notre exemple) et celles des pixels environnants (« N » dans l'exemple). Toutefois, le choix des valeurs de la matrice permet de spécifier la priorité de certains pixels pour le calcul de l'image résultante.

Par exemple, la matrice suivante, appliquée à un filtre Convolution, laissera l'image intacte, exactement comme l'image originale :

```
0   0   0
0   1   0
0   0   0
```

L'image n'a pas été modifiée car la valeur du pixel original a une priorité relative de 1 pour déterminer la couleur du pixel final, alors que les pixels environnants ont une priorité relative de 0 (autrement dit, leur couleur n'affecte pas l'image finale).

De même, la matrice suivante provoque un décalage à gauche de tous les pixels d'une image :

```
0   0   0
0   0   1
0   0   0
```

Notez que dans cet exemple, le pixel lui-même n'a aucun effet sur la valeur finale du pixel affiché au même emplacement dans l'image finale : seule la valeur du pixel de droite est utilisée pour déterminer la valeur résultante de chaque pixel.

En ActionScript, la matrice est une combinaison d'une occurrence de l'objet Array contenant les valeurs et deux propriétés spécifiant le nombre de lignes et de colonnes de la matrice. L'exemple suivant charge une image, puis lui applique un filtre Convolution sur la base de la matrice du listing précédent :

```
// 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);
```

Un point important n'est pas évident dans ce code : l'effet de valeurs autres que 1 ou 0 dans la matrice. Par exemple, la même matrice avec le chiffre 8 au lieu de 1 dans le coin supérieur droit effectuerait la même action (décalage des pixels vers la gauche). Toutefois, les couleurs de l'image seraient 8 fois plus lumineuses. En effet, les valeurs finales de couleur des pixels sont calculées en multipliant les valeurs de la matrice par celles des couleurs originales des pixels, en additionnant ces valeurs, puis en les divisant par celle de la propriété `divisor` du filtre. Notez que, dans cet exemple, la propriété `divisor` a la valeur 1. En règle générale, pour que la luminosité des couleurs reste à peu près identique à celle des couleurs de l'image originale, la propriété divisor doit avoir une valeur égale à la somme des valeurs de la matrice. Ainsi, avec une matrice dont la somme des valeurs est 8, et pour un diviseur de 1, l'image résultante sera environ 8 fois plus lumineuse que l'image originale.

Bien que l'effet de cette matrice ne soit pas très spectaculaire, d'autres valeurs peuvent être utilisées pour créer divers effets. Voici quelques ensembles standard de valeurs de matrice permettant d'obtenir divers effets avec une matrice de trois sur trois :

- Flou de base (diviseur 5) :

```
0 1 0
1 1 1
0 1 0
```

- Accentuation (diviseur 1) :

```
 0, -1, 0
-1, 5, -1
 0, -1, 0
```

- Détection des contours (diviseur 1) :

```
 0, -1, 0
-1, 4, -1
 0, -1, 0
```

- Estampage (diviseur 1) :

```
-2, -1, 0
-1, 1, 1
 0, 1, 2
```

Notez qu'avec la plupart de ces effets, le diviseur est 1. En effet, l'addition des valeurs négatives et des valeurs positives dans la matrice donne 1 (ou 0 dans le cas de la détection des contours, mais la valeur de la propriété `divisor` ne peut pas être 0).

Filtre Mappage de déplacement

La classe DisplacementMapFilter utilise des valeurs de pixel extraites d'un objet BitmapData (appelé image de mappage du déplacement) pour appliquer un effet de déplacement à un nouvel objet. L'image de mappage du déplacement est en général différente de l'occurrence d'objet d'affichage ou BitmapData à laquelle le filtre est appliqué. L'effet de déplacement nécessite de déplacer les pixels de l'image filtrée, autrement dit de les décaler d'un certain niveau. Ce filtre permet de créer un effet de décalage, de gondole ou de marbrure.

La direction et la valeur du déplacement appliqué à un pixel donné sont déterminées par la valeur colorimétrique de l'image de mappage du déplacement. Pour utiliser ce filtre, il est nécessaire de spécifier l'image de mappage, ainsi que les valeurs suivantes, qui permettent de contrôler le calcul du déplacement :

- Point de mappage : emplacement, dans l'image filtrée, auquel le coin supérieur gauche du filtre de déplacement sera appliqué. Ce paramètre n'est nécessaire que pour appliquer le filtre à une partie de l'image seulement.
- Composant X : canal couleur de l'image de mappage qui affecte la position x des pixels.
- Composant Y : canal couleur de l'image de mappage qui affecte la position y des pixels.
- Echelle X : valeur multiplicatrice qui indique le niveau de déplacement sur l'axe x.
- Echelle Y : valeur multiplicatrice qui indique le niveau de déplacement sur l'axe y.
- Mode de filtrage : détermine ce que Flash Player ou AIR doit faire dans le cas d'espaces vides créés par le décalage des pixels. Les options, définies par des constantes dans la classe DisplacementMapFilterMode, sont d'afficher les pixels originaux (mode `IGNORE`), de décaler et transférer les pixels de l'autre côté de l'image (mode `WRAP`, qui est le mode par défaut), d'utiliser le pixel déplacé le plus proche (mode `CLAMP`) ou de remplir ces espaces vides avec une couleur (mode `COLOR`).

Pour comprendre le fonctionnement d'un filtre de mappage de déplacement, prenons un exemple simple. Dans le code ci-dessous, une image est chargée, puis elle est centrée sur la scène et un filtre Mappage de déplacement lui est appliqué, ce qui déplace horizontalement (vers la gauche) les pixels de toute l'image.

```
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);
```

Les propriétés utilisées pour définir le déplacement sont les suivantes :

- Bitmap de mappage : le bitmap de déplacement est une nouvelle occurrence de BitmapData, créée par code. Ses dimensions sont identiques à celles de l'image chargée (le déplacement est donc appliqué à toute l'image). Elle est remplie de pixels rouges opaques.
- Point de mappage : cette valeur est définie pour le point 0, 0 (ici encore, le déplacement sera appliqué à toute l'image).
- Composant X : cette valeur reçoit la constante `BitmapDataChannel.RED`, ce qui signifie que c'est la valeur de rouge de l'image bitmap de mappage qui déterminera le niveau de déplacement des pixels sur l'axe x.
- Echelle X : cette valeur est réglée sur 250. L'image de mappage étant entièrement rouge, la valeur totale de déplacement ne décale l'image que faiblement (environ un demi-pixel). Si cette valeur était de 1, l'image ne serait donc décalée que de 0,5 pixel horizontalement. En choisissant une valeur de 250, nous déplaçons l'image d'environ 125 pixels.

Ces valeurs provoquent un déplacement des pixels de l'image filtrée de 250 pixels à gauche. La direction (gauche ou droite) et l'importance du déplacement dépendent de la valeur colorimétrique des pixels de l'image de mappage. Le principe de fonctionnement de ce filtre est le suivant : Flash Player ou AIR analyse un par un les pixels de l'image filtrée (ou tout au moins ceux de la zone à laquelle le filtre est appliqué, ce qui ici signifie toute l'image), et procède comme suit pour chaque pixel :

1. Il détermine le pixel correspondant dans l'image de mappage. Par exemple, pour calculer la valeur de déplacement du pixel du coin supérieur gauche de l'image filtrée, Flash Player ou AIR analyse le pixel correspondant dans le coin supérieur gauche de l'image de mappage.
2. Il détermine la valeur du canal de couleur spécifié dans le pixel de mappage. Dans cet exemple, le canal de couleur du composant x est le rouge. Flash Player et AIR recherchent donc la valeur du canal rouge au point en question dans l'image de mappage. L'image de mappage étant un rouge opaque, le canal rouge du pixel a la valeur 0xFF, soit 255. Cette valeur est la valeur de déplacement.
3. Ils comparent ensuite la valeur de déplacement à la valeur médiane (127, à mi-chemin entre 0 et 255). Si la valeur de déplacement est inférieure à la valeur médiane, le pixel est déplacé dans une direction positive (vers la droite pour l'axe x, vers le bas pour l'axe y). Par contre, si la valeur de déplacement est supérieure à la valeur médiane (comme dans cet exemple), le pixel est déplacé dans une direction négative (vers la gauche pour l'axe x, vers le haut pour l'axe y). Plus précisément, Flash Player et AIR soustraient la valeur de déplacement de 127, et le résultat (positif ou négatif) est la valeur relative de déplacement qui est appliquée.
4. Enfin, ils déterminent la valeur réelle de déplacement en calculant le pourcentage de déplacement complet représenté par la valeur de déplacement relatif. Dans cet exemple, un rouge à 100 % provoque un déplacement de 100 %. Ce pourcentage est ensuite multiplié par la valeur de l'échelle x ou de l'échelle y pour déterminer le nombre de pixels de déplacement à appliquer. Dans cet exemple, la valeur de déplacement est 100 % multiplié par un multiple de 250, soit environ 125 pixels à gauche.

Comme nous ne spécifions aucune valeur pour les composants x et y, les valeurs par défaut (qui ne provoquent pas de déplacement) sont utilisées. C'est pourquoi l'image n'est pas déplacée dans le sens vertical.

Dans cet exemple, le paramètre par défaut de mode de filtrage, `WRAP`, est utilisé, si bien que lorsque les pixels sont déplacés vers la gauche, l'espace laissé vide à droite est rempli par les pixels qui ont été décalés le long du côté gauche de l'image. Vous pouvez modifier cette valeur pour voir les différents effets ainsi obtenus. Par exemple, si vous ajoutez la ligne suivante dans la partie du code qui définit les propriétés de déplacement (avant la ligne `loader.filters = [displacementMap]`), l'image semble avoir subi un balayage :

```
displacementMap.mode = DisplacementMapFilterMode.CLAMP;
```

Le code ci-dessous propose un exemple plus complexe, en utilisant un filtre Mappage de déplacement pour créer un effet de loupe dans l'image :

```
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);
```

Ce code génère d'abord deux cercles en dégradé, qui sont combinés pour former l'image de mappage du déplacement. Le cercle rouge est à l'origine du déplacement sur l'axe x (`xyFilter.componentX = BitmapDataChannel.RED`) et le cercle bleu est à l'origine du déplacement sur l'axe y (`xyFilter.componentY = BitmapDataChannel.BLUE`). Pour vous permettre de comprendre plus aisément l'aspect de l'image de mappage du déplacement, le code affiche les cercles originaux, ainsi que le cercle combiné qui fait office d'image de mappage, en bas de l'écran.

Le code charge ensuite une image et, en fonction des déplacements de la souris, applique le filtre de déplacement à la partie de l'image qui est sous la souris. Les cercles dégradés utilisés pour l'image de mappage de déplacement provoquent un effet centrifuge dans la zone à laquelle le filtre est appliqué. Notez que les zones en gris de l'image de mappage du déplacement ne provoquent pas de déplacement. En effet, la valeur du gris est `0x7F7F7F`. Les canaux bleu et rouge de ce niveau de gris ont donc exactement une valeur médiane, si bien qu'il n'y a pas de déplacement lorsqu'une zone grise apparaît dans l'image de mappage. Il n'y a pas non plus de déplacement au centre du cercle. Bien que cette zone ne soit pas de couleur grise, ses canaux rouge et bleu ont des valeurs identiques à celles des canaux rouge et bleu du gris moyen, et puisque le déplacement est basé sur les valeurs de bleu et de rouge, aucun déplacement n'a lieu.

Filtre Shader

La classe ShaderFilter vous permet d'utiliser un effet de filtre personnalisé défini en tant que shader de Pixel Bender. Parce que l'effet de filtre est écrit en tant que shader de Pixel Bender, il peut être entièrement personnalisé. Le contenu filtré est transmis au shader en tant qu'image d'entrée et le résultat de l'opération du shader devient le résultat du filtre.

Remarque : *le filtre Shader est pris en charge dans ActionScript à partir de Flash Player 10 et Adobe AIR 1.5.*

Pour appliquer un filtre Shader à un objet, vous devez commencer par créer une occurrence de Shader représentant le shader de Pixel Bender en cours d'utilisation. Pour plus de détails concernant la procédure de création d'une occurrence de Shader et la définition d'une image d'entrée et des paramètres correspondants, consultez le chapitre « Utilisation des shaders de Pixel Bender » à la page 395.

Lorsque vous utilisez un shader en tant que filtre, vous devez tenir compte de trois considérations importantes :

- Le shader doit être défini pour accepter au moins une image d'entrée.
- L'objet filtré (objet d'affichage ou objet BitmapData auquel est appliqué le filtre) est transmis au shader en tant que première valeur d'image d'entrée. De ce fait, il est recommandé de ne pas définir manuellement la valeur de la première image d'entrée.

- Si le shader définit plusieurs images d'entrée, les autres entrées doivent être stipulées manuellement (en définissant la propriété `input` de toute occurrence de ShaderInput appartenant à l'occurrence de Shader).

Lorsque vous disposez d'un objet Shader pour le shader, vous créez une occurrence de ShaderFilter. Il s'agit de l'objet filtre en tant que tel utilisé comme tout autre filtre. Pour créer un élément ShaderFilter qui utilise un objet Shader, appelez le constructeur `ShaderFilter()` et transmettez l'objet Shader en tant qu'argument, comme illustré dans le code suivant :

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
```

Pour disposer d'un exemple complet d'utilisation d'un filtre Shader, consultez la section « Utilisation d'un shader comme filtre » à la page 413.

Exemple : Filter Workbench

L'exemple Filter Workbench comporte une interface utilisateur qui permet d'appliquer divers filtres à des images et autre contenu visuel et de voir le code résultant, qui peut être utilisé pour générer le même effet en ActionScript. Non seulement cette application fournit un outil permettant d'expérimenter avec les filtres, mais elle illustre également les techniques suivantes :

- Création d'occurrences de filtres divers
- Application de plusieurs filtres à un objet d'affichage

Pour obtenir les fichiers d'application associés à cet exemple, voir www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application Filter Workbench résident dans le dossier Samples/FilterWorkbench. L'application se compose des fichiers suivants :

Fichier	Description
com/example/programmingas3/filterWorkbench/FilterWorkbenchController.as	Classe qui fournit la principale fonctionnalité de l'application, notamment permuter le contenu auquel sont appliqués les filtres et appliquer les filtres au contenu.
com/example/programmingas3/filterWorkbench/IFilterFactory.as	Interface qui définit les méthodes courantes et implémentées par chacune des classes usine de filtres. Cette interface définit la fonctionnalité commune utilisée par la classe FilterWorkbenchController pour interagir avec chaque classe usine de filtres.
Dans le dossier com/example/programmingas3/filterWorkbench/ : BevelFactory.as BlurFactory.as ColorMatrixFactory.as ConvolutionFactory.as DropShadowFactory.as GlowFactory.as GradientBevelFactory.as GradientGlowFactory.as	Jeu de classes, dont chacune implémente l'interface IFilterFactory. Chacune de ces classes permet de créer et de définir les valeurs associées à un type unique de filtre. Les panneaux de propriétés des filtres de l'application utilisent ces classes usine pour créer des occurrences des filtres correspondants, que la classe FilterWorkbenchController extrait et applique au contenu d'image.

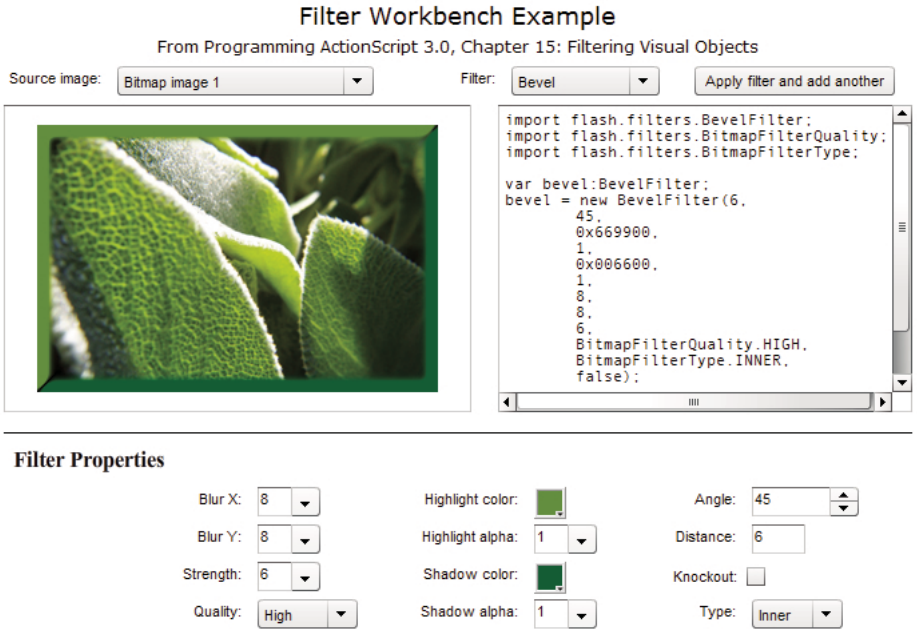
Fichier	Description
com/example/programmingas3/filterWorkbench/IFilterPanel.as	Interface qui définit les méthodes courantes et implémentées par les classes qui spécifient les panneaux de l'interface utilisateur employés pour manipuler les valeurs de filtre dans l'application.
com/example/programmingas3/filterWorkbench/ColorStringFormatter.as	Classe d'utilitaires qui comporte une méthode de conversion d'une valeur de couleur numérique au format chaîne hexadécimal.
com/example/programmingas3/filterWorkbench/GradientColor.as	Classe servant d'objet de valeur, qui combine en un objet unique les trois valeurs (couleur, alpha et rapport) associées à chaque couleur dans GradientBevelFilter et GradientGlowFilter.
Interface utilisateur (Flash)	
FilterWorkbench.fla	Fichier principal qui définit l'interface utilisateur de l'application.
flashapp/FilterWorkbench.as	Classe qui assure la fonctionnalité de l'interface utilisateur de l'application principale. Elle sert de classe de document au fichier FLA de l'application.
Dans le dossier flashapp/filterPanels : BevelPanel.as BlurPanel.as ColorMatrixPanel.as ConvolutionPanel.as DropShadowPanel.as GlowPanel.as GradientBevelPanel.as GradientGlowPanel.as	Jeu de classes qui assurent la fonctionnalité de chaque panneau utilisé pour définir les options d'un filtre unique. A chaque classe correspond également un symbole MovieClip dans la bibliothèque du fichier FLA de l'application principale, dont le nom est identique à celui de la classe (par exemple, le symbole « BlurPanel » est lié à la classe définie dans BlurPanel.as). Les composants de l'interface utilisateur sont placés et identifiés par nom dans ces symboles.
flashapp/ImageContainer.as	Objet d'affichage qui sert de conteneur à l'image chargée à l'écran.
flashapp/BGColorCellRenderer.as	Composant de rendu de cellule personnalisé permettant de modifier la couleur d'arrière-plan d'une cellule dans le composant DataGrid
flashapp/ButtonCellRenderer.as	Composant de rendu de cellule personnalisé permettant d'insérer un composant Button dans une cellule du composant DataGrid
Contenu d'image filtré	
com/example/programmingas3/filterWorkbench/ImageType.as	Cette classe sert d'objet de valeur contenant le type et l'URL d'un fichier d'image unique, dans lequel l'application peut charger et appliquer des filtres. Cette classe comporte également un jeu de constantes qui représentent les fichiers d'image en tant que tels disponibles.
images/sampleAnimation.swf, images/sampleImage1.jpg, images/sampleImage2.jpg	Images et autre contenu visuel auxquels sont appliqués des filtres dans l'application.

Utilisation des filtres ActionScript

L'application Filter Workbench est conçue pour vous aider à expérimenter avec divers effets de filtre et générer le code ActionScript approprié correspondant. Elle vous permet de sélectionner trois fichiers distincts comportant un contenu visuel, tel que des images bitmap et une animation créée par Flash et d'appliquer huit filtres ActionScript distincts à l'image sélectionnée, soit seuls, soit combinés à d'autres filtres. L'application comprend les filtres suivants :

- Biseau (flash.filters.BevelFilter)
- Flou (flash.filters.BlurFilter)
- Matrice de couleurs (flash.filters.ColorMatrixFilter)
- Convolution (flash.filters.ConvolutionFilter)
- Ombre portée (flash.filters.DropShadowFilter)
- Rayonnement (flash.filters.GlowFilter)
- Biseau dégradé (flash.filters.GradientBevelFilter)
- Rayonnement dégradé (flash.filters.GradientGlowFilter)

Lorsqu'un utilisateur a sélectionné une image et un filtre à appliquer à celle-ci, l'application affiche un panneau contenant des contrôles de définition des propriétés du filtre sélectionné. Par exemple, l'image suivante illustre l'application dans laquelle est sélectionné le filtre Biseau :



Lorsque l'utilisateur règle les propriétés du filtre, l'aperçu est actualisé en temps réel. L'utilisateur peut également appliquer plusieurs filtres. Pour ce faire, il personnalise un filtre, clique sur le bouton Apply, personnalise un autre filtre, clique sur le bouton Apply, et ainsi de suite.

Les panneaux de filtre de l'application proposent diverses fonctions et sont soumis à quelques limites :

- Le filtre Matrice de couleurs comprend un jeu de contrôles permettant de manipuler directement des propriétés d'image courantes telles que la luminosité, les contrastes, la saturation et la teinte. Il est également possible de définir des valeurs de matrice de couleurs personnalisées.

- Le filtre Convolution, qui n'est disponible qu'en ActionScript, comprend un jeu de valeurs de matrice de convolution couramment utilisées. Il est également possible de définir des valeurs personnalisées. Toutefois, bien que la classe ConvolutionFilter gère une matrice de n'importe quelle taille, l'application Filter Workbench utilise une matrice de 3 x 3 fixe, soit la taille de filtre la plus fréquemment utilisée.
- Le filtre Mappage de déplacement et le filtre Shader, réservés à ActionScript, ne sont pas disponibles dans l'application Filter Workbench. Outre le contenu d'image filtré, un filtre Mappage de déplacement nécessite une image de mappage. L'image de mappage utilisée par le filtre Mappage de déplacement a un impact majeur sur le résultat du filtre. Puisqu'il est impossible de charger ou créer une image de mappage, il s'avérerait très difficile d'expérimenter avec le filtre Mappage de déplacement. De même, outre le contenu d'image filtré, un filtre Shader requiert un fichier de pseudo-code binaire Pixel Bender. Puisqu'il est impossible de charger le pseudo-code binaire du shader, il est impossible d'utiliser un filtre Shader.

Création d'occurrences de filtre

L'application Filter Workbench comprend un jeu de classes, une par filtre disponible, qui sont utilisées par les divers panneaux pour créer les filtres. Lorsqu'un utilisateur sélectionne un filtre, le code ActionScript associé au panneau de filtre crée une occurrence de la classe de filtres usine appropriée. (Ces classes portent le nom de *classes usine*, car elles ont pour objet de créer des occurrences d'autres objets, à l'instar d'une vraie usine qui fabrique des produits).

Lorsque l'utilisateur modifie la valeur d'une propriété dans le panneau, le code correspondant appelle la méthode appropriée dans la classe usine. Chaque classe usine comporte des méthodes spécifiques utilisées par le panneau pour créer l'occurrence de filtre appropriée. Par exemple, si l'utilisateur sélectionne le filtre Flou, l'application crée une occurrence de BlurFactory. La classe BlurFactory comporte une méthode `modifyFilter()` qui gère trois paramètres : `blurX`, `blurY` et `quality`. Conjointement, ces paramètres permettent de créer l'occurrence de BlurFilter requise :

```
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));
}
```

En revanche, si l'utilisateur sélectionne le filtre Convolution, celui-ci étant beaucoup plus souple, il contrôle un nombre supérieur de propriétés. Dans la classe ConvolutionFactory, le code suivant est appelé lorsque l'utilisateur sélectionne une autre valeur dans le panneau des filtres :

```
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));
}
```

Notez que dans chaque cas, lorsque les valeurs du filtre sont modifiées, l'objet usine distribue un événement `Event.CHANGE` pour avertir les écouteurs de la modification. La classe FilterWorkbenchController, qui applique les filtres au contenu filtré, recherche cet événement pour s'assurer qu'elle doit extraire une nouvelle copie du filtre et l'appliquer à nouveau au contenu filtré.

La classe FilterWorkbenchController n'a pas besoin de connaître les détails précis de chaque classe usine associé au filtre. Elle doit juste savoir que le filtre a été modifié et être en mesure d'accéder à une copie de ce dernier. A cet effet, l'application comporte une interface, IFilterFactory, qui définit le comportement requis d'une classe usine de filtres pour que l'occurrence de FilterWorkbenchController de l'application soit en mesure de fonctionner correctement. L'interface IFilterFactory définit la méthode `getFilter()` utilisée par la classe FilterWorkbenchController :

```
function getFilter():BitmapFilter;
```

Notez que la définition de la méthode de l'interface `getFilter()` stipule de renvoyer une occurrence de BitmapFilter plutôt qu'un type déterminé de filtre. La classe BitmapFilter ne définit pas de type spécifique de filtre. BitmapFilter constitue de fait la classe de base sur laquelle se fondent toutes les classes de filtre. Chaque classe usine de filtres définit une implémentation déterminée de la méthode `getFilter()`, dans laquelle elle renvoie une référence à l'objet filtre généré. Par exemple, une version abrégée du code source de la classe ConvolutionFactory est illustrée ci-après :

```
public class ConvolutionFactory extends EventDispatcher implements IFilterFactory
{
    // ------- Private vars -------
    private var _filter:ConvolutionFilter;
    ...
    // ------- IFilterFactory implementation -------
    public function getFilter():BitmapFilter
    {
        return _filter;
    }
    ...
}
```

Dans l'implémentation de la classe ConvolutionFactory de la méthode `getFilter()`, elle renvoie une occurrence de ConvolutionFilter, bien que tout objet qui appelle `getFilter()` ne sache pas nécessairement que, conformément à la définition de la méthode `getFilter()` appliquée par ConvolutionFactory, il doit renvoyer toute occurrence de BitmapFilter, soit une occurrence de n'importe quelle classe de filtre ActionScript.

Application de filtres aux objets d'affichage

Comme indiqué dans la section précédente, l'application Filter Workbench utilise une occurrence de la classe FilterWorkbenchController (appelée à partir de maintenant l'« occurrence de contrôleur »), chargée d'appliquer les filtres à l'objet visuel sélectionné. Avant que l'occurrence de contrôleur ne puisse appliquer un filtre, elle doit identifier l'image ou le contenu visuel concerné. Lorsque l'utilisateur sélectionne une image, l'application appelle la méthode `setFilterTarget()` de la classe FilterWorkbenchController et transmet l'une des constantes définies dans la classe ImageType :

```
public function setFilterTarget(targetType:ImageType):void
{
    ...
    _loader = new Loader();
    ...
    _loader.contentLoaderInfo.addEventListener(Event.COMPLETE, targetLoadComplete);
    ...
}
```

En se référant à ces informations, l'occurrence de contrôleur charge le fichier indiqué, puis le stocke dans une variable d'occurrence appelée `_currentTarget` :

```
private var _currentTarget:DisplayObject;

private function targetLoadComplete(event:Event):void
{
    ...
    _currentTarget = _loader.content;
    ...
}
```

Lorsque l'utilisateur sélectionne un filtre, l'application appelle la méthode `setFilter()` de l'occurrence de contrôleur en indiquant au contrôleur une référence à l'objet Filter Factory approprié, qu'elle stocke dans une variable d'occurrence appelée `_filterFactory`.

```
private var _filterFactory:IFilterFactory;

public function setFilter(factory:IFilterFactory):void
{
    ...

    _filterFactory = factory;
    _filterFactory.addEventListener(Event.CHANGE, filterChange);
}
```

Notez que, comme indiqué précédemment, l'occurrence de contrôleur ne connaît pas le type de données précis de l'occurrence de Filter Factory assigné. Elle ne sait qu'une chose, c'est que l'objet implémente l'occurrence de IFilterFactory, ce qui signifie qu'il possède une méthode `getFilter()` et distribue un événement `change` (`Event.CHANGE`) lorsque le filtre est modifié.

Lorsque l'utilisateur modifie les propriétés d'un filtre dans le panneau des filtres, l'occurrence de contrôleur est avertie de la modification par l'événement `change` de Filter Factory, qui appelle la méthode `filterChange()` de l'occurrence de contrôleur. Cette méthode appelle alors la méthode `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();
}
```

L'application du filtre à l'objet d'affichage se produit au sein de la méthode `applyTemporaryFilter()`. Le contrôleur extrait d'abord une référence à l'objet filtre en appelant la méthode `getFilter()` de Filter Factory.

```
var currentFilter:BitmapFilter = _filterFactory.getFilter();
```

L'occurrence de contrôleur possède une variable d'occurrence de Array appelée `_currentFilters`, dans laquelle elle stocke tous les filtres appliqués à l'objet d'affichage. L'étape suivante consiste à ajouter le filtre mis à jour à ce tableau :

```
_currentFilters.push(currentFilter);
```

Le code assigne ensuite le tableau de filtres à la propriété `filters` de l'objet d'affichage, qui applique à proprement parler les filtres à l'image :

```
_currentTarget.filters = _currentFilters;
```

Enfin, puisque le filtre appliqué en dernier demeure le filtre de « travail », il ne doit pas être appliqué à titre définitif à l'objet d'affichage. Il est donc supprimé du tableau `_currentFilters` :

```
_currentFilters.pop();
```

Supprimer ce filtre du tableau n'affecte pas l'objet d'affichage filtré, car un objet d'affichage effectue une copie du tableau de filtres lorsqu'il est assigné à la propriété `filters` et utilise ce tableau interne au lieu du tableau initial. De ce fait, toute modification du tableau de filtres n'affecte pas l'objet d'affichage tant que le tableau n'a pas été à nouveau assigné à la propriété `filters` de l'objet d'affichage.

Chapitre 17 : Utilisation des shaders de Pixel Bender

Adobe Pixel Bender Toolkit permet aux développeurs de composer des shaders qui créent des effets graphiques et autres. Le pseudo-code binaire Pixel Bender peut être exécuté dans ActionScript pour appliquer l'effet aux données de l'image ou au contenu visuel. L'utilisation des shaders de Pixel Bender vous donne la possibilité de créer des effets visuels personnalisés et de traiter des données au-delà des fonctions incorporées à ActionScript.

Remarque : *Pixel Bender est pris en charge depuis Flash Player version 10 et Adobe AIR version 1.5.*

Principes de base des shaders de Pixel Bender

Introduction à l'utilisation des shaders de Pixel Bender

Adobe Pixel Bender est un langage de programmation qui permet de créer ou de manipuler un contenu d'image. Par le biais de Pixel Bender, vous créez un noyau, également appelé dans ce document un shader. Le shader définit une fonction unique exécutée séparément sur chacun des pixels d'une image. Le résultat de chaque appel à la fonction est la couleur de sortie aux coordonnées de ce pixel dans l'image. Vous pouvez spécifier des valeurs de paramètre et images d'entrée pour personnaliser l'opération. Si un shader est exécuté une seule fois, les valeurs de paramètres et d'entrée sont des constantes. Les seuls éléments qui varient sont les coordonnées du pixel dont la couleur est le résultat de l'appel de la fonction.

Dans la mesure du possible, la fonction shader est appelée pour plusieurs coordonnées de pixel de sortie en parallèle. Les performances du shader sont ainsi optimisées et le traitement s'avère souvent particulièrement efficace.

Dans Flash Player et Adobe AIR, l'utilisation d'un shader permet de créer aisément trois types d'effets :

- remplissage d'un dessin
- mode de fondu
- filtre

Il est également possible d'exécuter un shader en mode autonome. En mode autonome, vous accédez directement au résultat d'un shader au lieu de prédéfinir l'usage prévu. Le résultat correspond à des données d'image, des données binaires ou des données numériques. Il n'est pas nécessaire que les données soient des données d'image. Vous pouvez ainsi affecter à un shader un jeu de données en entrée. Le shader traite les données et vous accédez au résultat qu'il renvoie.

Remarque : *Pixel Bender est pris en charge depuis Flash Player version 10 et Adobe AIR version 1.5.*

Tâches courantes des shaders de Pixel Bender

Les opérations suivantes, qui sont décrites dans ce chapitre, peuvent être aisément accomplies en ActionScript à l'aide de filtres :

- Chargement d'un shader dans une application SWF en cours d'exécution ou intégration du shader lors de la compilation, auquel vous accédez lors de l'exécution
- Accès aux métadonnées du shader

- Identification et définition des valeurs des entrées du shader (il s'agit en règle générale d'images)
- Identification et définition des valeurs des paramètres du shader
- Utilisation d'un shader comme suit :
  - En tant que remplissage d'un dessin
  - En tant que mode de fondu
  - En tant que filtre
  - En mode autonome

Terminologie et concepts importants

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Noyau : pour Pixel Bender, un noyau est équivalent à un shader. Par le biais de Pixel Bender, votre code définit un noyau, qui définit une fonction unique exécutée séparément sur chacun des pixels d'une image.
- Pseudo-code binaire Pixel Bender : lorsqu'un noyau Pixel Bender est compilé, il est transformé en pseudo-code binaire Pixel Bender. Flash Player ou Adobe AIR accède au pseudo-code binaire et l'exécute lors de l'exécution.
- Langage Pixel Bender : langage de programmation utilisé pour créer un noyau Pixel Bender.
- Pixel Bender Toolkit : application utilisée pour créer un fichier de pseudo-code binaire Pixel Bender à partir du code source Pixel Bender. Toolkit vous permet d'écrire, de tester et de compiler un code source Pixel Bender.
- Shader : dans le cadre de ce document, un shader est un jeu de fonctionnalités écrites dans le langage Pixel Bender. Le code d'un shader crée un effet visuel ou effectue un calcul. Dans les deux cas, le shader renvoie un jeu de données (il s'agit en règle générale des pixels d'une image). Le shader exécutant la même opération sur chaque point de données, seules les coordonnées du pixel de sortie varient.

  Le shader n'est pas écrit en ActionScript. Il est écrit dans le langage Pixel Bender et compilé en pseudo-code binaire Pixel Bender. Il peut être intégré à un fichier SWF lors de la compilation ou chargé en tant que fichier externe lors de l'exécution. Dans les deux cas, pour y accéder dans ActionScript, il est nécessaire de créer un objet Shader et de le lier au pseudo-code binaire du shader.
- Entrée du shader : entrée complexe, généralement composée de données d'image bitmap, fournie à un shader qui l'utilise dans ses calculs. Pour chaque variable d'entrée définie dans un shader, une valeur unique (une image unique ou un jeu de données binaires) est utilisée pour l'exécution de bout en bout du shader.
- Paramètre de shader : valeur unique (ou jeu de valeurs limité) fournie à un shader, qui l'utilise dans ses calculs. Chaque valeur de paramètre est définie pour une exécution de shader unique et la même valeur est utilisée de bout en bout lors de l'exécution du shader.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester les exemples de code fournis. Etant donné que ce chapitre décrit comment créer et manipuler du contenu visuel, le test du code implique l'exécution du code et l'affichage des résultats dans le fichier SWF créé. Tous les exemples créent un contenu par le biais de l'API de dessin qui utilise l'effet de shader ou est modifiée par ce dernier.

La plupart des exemples de code se composent de deux parties. La première partie correspond au code source Pixel Bender associé au shader utilisé dans l'exemple. Vous devez commencer par utiliser Pixel Bender Toolkit pour compiler le code source dans un fichier de pseudo-code binaire Pixel Bender. Procédez comme suit pour créer le fichier de pseudo-code binaire Pixel Bender :

1. Ouvrez Adobe Pixel Bender Toolkit. Le cas échéant, dans le menu Build (Développement), choisissez « Turn on Flash Player warnings and errors » (Activer les avertissements et erreurs Flash Player).
2. Copiez le code Pixel Bender et collez-le dans le panneau de l'éditeur de code de Pixel Bender Toolkit.
3. Dans le menu File (Fichier), choisissez « Export kernel filter for Flash Player » (Exporter le filtre de noyau associé à Flash Player).
4. Enregistrez le fichier de pseudo-code binaire Pixel Bender dans le même répertoire que le document Flash. Le nom du fichier doit être identique à celui stipulé dans la description de l'exemple.

La partie ActionScript de chaque exemple est écrite en tant que fichier de classe. Pour tester l'exemple :

1. Créez un document Flash vide et enregistrez-le sur votre ordinateur.
2. Créez un nouveau fichier ActionScript et enregistrez-le dans le même répertoire que le document Flash. Le nom du fichier doit correspondre au nom de la classe du code. Par exemple, si le code définit une classe MyApplication, enregistrez le fichier ActionScript sous le nom MyApplication.as.
3. Copiez le code dans le fichier ActionScript et enregistrez le fichier.
4. Dans le document Flash, cliquez sur une partie vide de la scène ou de l'espace de travail pour activer l'Inspecteur des propriétés du document.
5. Dans l'Inspecteur des propriétés, dans le champ Classe du document, saisissez le nom de la classe ActionScript que vous avez copiée du texte.
6. Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

   Les résultats de l'exemple s'affichent dans la fenêtre d'aperçu.

Ces techniques de test d'exemples de code sont expliquées plus en détail à la section « Test des exemples de code contenus dans un chapitre » à la page 36.

Chargement ou intégration d'un shader

La première étape dans l'utilisation d'un shader de Pixel Bender dans ActionScript consiste à pouvoir accéder au shader en code ActionScript. Comme un shader est créé à l'aide d'Adobe Pixel Bender Toolkit et rédigé dans le langage Pixel Bender, on ne peut pas y accéder directement dans ActionScript. Il faut plutôt créer une occurrence de la classe Shader qui représente le shader de Pixel Bender à ActionScript. L'objet Shader vous permet de trouver les informations concernant le shader : par exemple, s'il prévoit de recevoir des paramètres ou des valeurs pour l'image d'entrée. Vous passez l'objet Shader aux autres objets pour utiliser effectivement le shader. Par exemple, pour utiliser le shader comme filtre, vous affectez l'objet Shader à la propriété `shader` d'un objet ShaderFilter. Ou bien, afin d'utiliser le shader pour remplir l'écran dans un dessin, vous passez l'objet Shader comme argument à la méthode `Graphics.beginShaderFill()`.

Votre code ActionScript peut accéder à un shader créé par Adobe Pixel Bender Toolkit (un fichier .pbj) de deux façons distinctes :

- Chargement lors de l'exécution : le fichier shader peut être chargé comme un élément externe à l'aide d'un objet URLLoader. Cette technique est comparable au chargement d'un élément externe, comme un fichier texte, par exemple. L'exemple suivant montre le chargement à l'exécution d'un fichier de pseudo-code binaire de shader et sa liaison à une occurrence de Shader :

```
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
}
```

- Intégration au fichier SWF : le fichier shader peut être intégré au fichier SWF lors de la compilation à l'aide de la balise de métadonnées `[Embed]`. La balise de métadonnées `[Embed]` n'est disponible que si vous utilisez le kit de développement Flex pour compiler le fichier SWF. Le paramètre `source` de la balise `[Embed]` pointe vers le fichier du shader et son paramètre `mimeType` est `"application/octet-stream"`, comme dans l'exemple ci-dessous :

```
[Embed(source="myShader.pbj", mimeType="application/octet-stream")]
var MyShaderClass:Class;

// ...

// create a shader and set the embedded shader as its bytecode
var shaderShader = 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
```

Dans les deux cas, vous liez le pseudo-code brut du shader (la propriété `URLLoader.data` ou une occurrence de la classe de données `[Embed]`) à l'occurrence de Shader. Comme le montrent les exemples précédents, vous pouvez affecter le pseudo-code binaire à l'occurrence du shader de deux façons. Vous pouvez transmettre le pseudo-code binaire du shader sous forme d'argument au constructeur `Shader()`. Vous pouvez également le définir en tant que propriété `byteCode` de l'occurrence de Shader.

Dès qu'un shader de Pixel Bender est créé et lié à l'objet Shader, vous pouvez utiliser le shader pour créer des effets de plusieurs façons. Vous pouvez l'utiliser en tant que filtre, mode de fondu, remplissage de bitmap ou comme moyen de traitement autonome d'image bitmap ou autres données. Vous pouvez également utiliser la propriété `data` de l'objet Shader pour accéder aux métadonnées du shader, spécifier les images d'entrée et spécifier les valeurs du paramétrage.

Accès aux métadonnées du shader

Tandis qu'il crée un noyau de shader de Pixel Bender, le programmeur peut spécifier des métadonnées sur le shader dans le code source Pixel Bender. Vous pouvez inspecter le shader et extraire ses métadonnées pendant que vous l'utilisez dans ActionScript.

Lorsque vous créez une occurrence de Shader et que vous la liez à un shader de Pixel Bender, un objet ShaderData, qui contient des données sur le shader, est créé et enregistré dans la propriété `data` de l'objet Shader. La classe ShaderData ne définit aucune de ses propres propriétés. Toutefois, lors de l'exécution, une propriété est ajoutée dynamiquement à l'objet ShaderData pour chaque valeur de métadonnées définie dans le code source du shader. Le nom attribué à chaque propriété est pareil à celui spécifié dans les métadonnées. Par exemple, supposez que le code source d'un shader de Pixel Bender contienne la définition des métadonnées suivante :

```
namespace : "Adobe::Example";
vendor : "Bob Jones";
version : 1;
description : "Creates a version of the specified image with the specified brightness.";
```

L'objet ShaderData créé pour ce shader l'est avec les propriétés et valeurs suivantes :

- `namespace` (String): `"Adobe::Example"`
- `vendor` (String): `"Bob Jones"`
- `version` (String): `"1"`
- `description` (String): `"Crée une version de l'image spécifiée avec la luminosité indiquée"`

Comme les propriétés de métadonnées sont ajoutées dynamiquement à l'objet ShaderData, vous pouvez utiliser une boucle `for..in` pour inspecter l'objet ShaderData. Cette technique vous permet de vous rendre compte si le shader possède des métadonnées et quelles sont ses valeurs. Outre les propriétés des métadonnées, un objet ShaderData peut avoir des propriétés représentant des entrées et des paramètres qui sont définis dans le shader. Lorsque vous utilisez une boucle `for..in` pour inspecter un objet ShaderData, vérifiez le type de données de chaque propriété pour vous rendre compte si la propriété est une entrée (une occurrence de ShaderInput), un paramètre (une occurrence de ShaderParameter) ou une valeur de métadonnées (une occurrence de String). L'exemple ci-dessous montre comment utiliser une boucle `for..in` pour examiner les propriétés dynamiques d'une propriété `data` d'un shader. Chaque valeur de métadonnées est ajoutée à une occurrence de Vector appelée `metadata`. Remarquez que cet exemple suppose qu'une occurrence de Shader appelée `myShader` existe déjà :

```
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
```

Pour une version de cet exemple, qui extrait également des entrées et des paramètres de shader, consultez la section « Identification des entrées et des paramètres d'un shader » à la page 400. Pour plus d'informations sur les propriétés des entrées et des paramètres, consultez la section « Spécification des valeurs des entrées et des paramètres d'un shader » à la page 400.

Spécification des valeurs des entrées et des paramètres d'un shader

De nombreux shaders de Pixel Bender sont définis pour faire appel à une ou plusieurs images utilisées au cours de son traitement. Par exemple, il est courant pour un shader d'accepter une image source et de la produire dotée d'un effet particulier. Suivant l'utilisation du shader, soit la valeur d'entrée est spécifiée automatiquement, soit il faut en fournir une. De la même façon, de nombreux shaders spécifient des paramètres utilisés dans l'individualisation de ce qu'ils produisent. Il faut également définir explicitement une valeur pour chaque paramètre avant d'utiliser le shader.

Vous utilisez la propriété `data` de l'objet Shader pour définir les entrées et les paramètres et pour établir si un shader en particulier prévoit des entrées et des paramètres. La propriété `data` est une occurrence de ShaderData.

Identification des entrées et des paramètres d'un shader

La première étape dans la spécification de valeurs d'entrée et de paramètres d'un shader consiste à savoir si celui que vous utilisez prévoit de recevoir des images d'entrée ou des paramètres. Chaque occurrence de Shader possède une propriété `data` qui contient un objet ShaderData. Si le shader spécifie des entrées ou des paramètres, on y accède en tant que propriétés de cet objet ShaderData. Les noms des propriétés correspondent aux noms spécifiés dans le code source du shader pour les entrées et les paramètres. Par exemple, si un shader spécifie une entrée appelée `src`, l'objet ShaderData possède une propriété appelée `src` qui représente cette entrée. Chaque propriété qui représente une entrée est une occurrence de ShaderInput et chaque propriété qui représente un paramètre est une occurrence de ShaderParameter.

Idéalement, le programmeur du shader fournit une documentation pour le shader afin de décrire quels sont les paramètres et les valeurs de l'image d'entrée qu'il prévoit, ce qu'ils représentent, les valeurs appropriées, et ainsi de suite.

Toutefois, si le shader n'est pas documenté et que vous ne disposez pas du code source, vous pouvez inspecter les données du shader pour identifier ces entrées et paramètres. Les propriétés qui représentent ces entrées et paramètres sont ajoutées dynamiquement à l'objet ShaderData. Par conséquent, vous pouvez utiliser une boucle `for..in` pour inspecter l'objet ShaderData afin de savoir si le shader qui lui est associé spécifie de tels entrées ou paramètres. Comme le décrit la section « Accès aux métadonnées du shader » à la page 399, on accède également, en tant que propriété dynamique ajoutée à la propriété `Shader.data`, à toute valeur de métadonnées définie pour un shader. Lorsque vous utilisez cette technique afin d'identifier entrées et paramètres, vérifiez le type de données des propriétés dynamiques. Si une propriété est une occurrence de ShaderInput, elle représente une entrée. S'il s'agit d'une occurrence de ShaderParameter, elle représente un paramètre. Sinon, il s'agit d'une valeur de métadonnées. L'exemple ci-dessous montre comment utiliser une boucle `for..in` pour inspecter les propriétés dynamiques d'une propriété `data` d'un shader. Chaque entrée (objet ShaderInput) est ajoutée à une occurrence de Vector appelée `inputs`. Chaque paramètre (objet ShaderParameter) est ajouté à une occurrence de Vector appelée `parameters`. Enfin, toute propriété de métadonnées est ajoutée à une occurrence de Vector appelée `metadata`. Remarquez que cet exemple suppose qu'une occurrence de Shader appelée `myShader` existe déjà :

```
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
```

Spécification des valeurs d'entrée d'un shader

De nombreux shaders prévoient de recevoir une ou plusieurs images d'entrée qui sont utilisées au cours de son traitement. Toutefois, dans nombre de cas, une entrée est automatiquement spécifiée lorsque l'objet Shader est utilisé. Par exemple, supposons qu'un shader ait besoin d'une entrée et qu'il serve de filtre. Lorsque le filtre est appliqué à un objet d'affichage ou BitmapData, cet objet est défini automatiquement comme entrée. Dans ce cas, on ne spécifie pas explicitement une valeur d'entrée.

Toutefois, dans certains cas, et plus particulièrement si un shader spécifie plusieurs entrées, il faut définir explicitement une valeur pour l'entrée. Toute entrée définie dans un shader est représentée dans ActionScript par un objet ShaderInput. L'objet ShaderInput est une propriété de l'occurrence de ShaderData dans la propriété `data` de l'objet Shader. Une description se trouve dans la section « Identification des entrées et des paramètres d'un shader » à la page 400. Par exemple, supposons qu'un shader définisse une entrée appelée `src` et que le shader soit lié à un objet Shader appelé `myShader`. A ce moment-là, on accède à l'objet ShaderInput qui correspond à l'entrée `src` à l'aide de l'identificateur suivant :

```
myShader.data.src
```

Chaque objet ShaderInput possède une propriété `input` qui est utilisée pour définir la valeur de l'entrée. On définit la propriété `input` sur une occurrence de BitmapData pour spécifier des données d'image. On peut aussi définir la propriété `input` sur BitmapData ou Vector.occurrence de <Number> pour spécifier des données binaires ou des nombres. Pour des détails et des informations sur les contraintes dans l'utilisation de BitmapData ou Vector.Pour l'occurrence de <Number> en tant qu'entrée, consultez le répertoire `ShaderInput.input` dans le Guide de référence du langage.

En plus de la propriété `input`, un objet ShaderInput possède des propriétés qui peuvent être utilisées pour déterminer quel type d'image prévoit l'entrée. Ces propriétés contiennent elles-mêmes les propriétés `width`, `height` et `channels`. Chaque objet ShaderInput possède également une propriété `index` qui est utile afin de déterminer si une valeur explicite doit être fournie pour l'entrée. Si un shader prévoit de recevoir davantage d'entrées que le nombre fixé automatiquement, vous pouvez alors définir des valeurs pour ces entrées. Pour plus de détails sur les différentes façons d'utiliser un shader et pour savoir si oui ou non les valeurs d'entrée sont fixées automatiquement, consultez la section « Utilisation d'un shader » à la page 405.

Spécification des valeurs de paramètres dans un shader

Certains shaders définissent des valeurs de paramètres qu'ils utilisent dans la création de leur résultat. Par exemple, un shader qui modifie la luminosité d'une image pourrait spécifier un paramètre de luminosité qui détermine dans quelle mesure l'opération affecte cette luminosité. Un paramètre unique défini dans un shader peut prévoir une ou plusieurs valeurs suivant la façon dont il a été spécifié dans le shader. Tout paramètre défini dans un shader est représenté dans ActionScript par un objet ShaderParameter. L'objet ShaderParameter est une propriété de l'occurrence de ShaderData dans la propriété des données de l'objet Shader. Une description se trouve dans la section « Identification des entrées et des paramètres d'un shader » à la page 400. Par exemple, supposons qu'un shader qui définit un paramètre appelé `luminosité` soit représenté par un objet Shader appelé `myShader`. Vous accédez alors à l'objet ShaderParameter correspondant au paramètre `luminosité` à l'aide de l'identificateur suivant :

```
myShader.data.brightness
```

Pour spécifier une ou plusieurs valeurs pour le paramètre, créez un tableau ActionScript contenant la ou les valeurs et affectez-le à la propriété `value` de l'objet ShaderParameter. La propriété `value` est définie en tant qu'occurrence de Array car il est possible qu'un seul paramètre du shader nécessite plusieurs valeurs. Même si le paramètre du shader ne prévoit qu'une seule valeur, il vous faut placer la valeur dans un objet Array pour l'affecter à la propriété `ShaderParameter.value`. Le code suivant montre comment définir une seule valeur pour la propriété `value`.

```
myShader.data.brightness.value = [75];
```

Si le code source Pixel Bender pour le shader spécifie une valeur par défaut pour le paramètre, un tableau contenant la ou les valeurs par défaut est créé et affecté à la propriété `value` de l'objet ShaderParameter lorsque l'objet Shader est créé. Dès qu'un tableau est affecté à la propriété `value`, même s'il s'agit de celui par défaut, la valeur du paramètre peut être modifiée en changeant la valeur de l'élément du tableau. Il n'est pas nécessaire de créer un autre tableau et de l'affecter à la propriété `value`.

L'exemple ci-dessous indique comment définir une valeur de paramètre dans un shader avec ActionScript. Dans cet exemple, le shader définit un paramètre appelé `color`. Ce paramètre `color` est déclaré en tant que variable `float4` dans le code source Pixel Bender, ce qui signifie qu'il s'agit d'un tableau à quatre nombres en virgule flottante. Dans l'exemple, la valeur du paramètre `color` change constamment. A chaque changement, le shader est utilisé pour dessiner un rectangle coloré à l'écran. Il en résulte un changement de couleur animé.

Remarque : *le code de cet exemple a été rédigé par Ryan Taylor. Merci Ryan de bien vouloir nous en faire profiter. Vous pouvez accéder aux exemples de Ryan et lire ses commentaires à l'adresse www.boostworthy.com/.*

Le code ActionScript repose sur l'utilisation de trois méthodes :

- `init()` : dans la méthode `init()`, le code charge le fichier de pseudo-code binaire Pixel Bender contenant le shader. Lors du chargement du fichier, la méthode `onLoadComplete()` est appelée.
- `onLoadComplete()` : dans la méthode `onLoadComplete()`, le code crée l'objet Shader appelé `shader`. Il crée également une occurrence de Sprite appelée `texture`. Dans la méthode `renderShader()`, le code dessine une fois par image le résultat du shader dans `texture`.
- `onEnterFrame()` : la méthode `onEnterFrame()` est appelée une fois par image, créant ainsi un effet d'animation. Dans cette méthode, le code définit la valeur du paramètre du shader sur la nouvelle couleur, puis appelle la méthode `renderShader()` pour dessiner le résultat du shader sous forme de rectangle.
- `renderShader()` : dans la méthode `renderShader()`, le code appelle la méthode `Graphics.beginShaderFill()` pour définir un remplissage de shader. Il dessine ensuite un rectangle dont le remplissage est défini par la sortie du shader (la couleur générée). Pour plus d'informations sur ce type d'utilisation d'un shader, consultez la section « Utilisation d'un shader comme outil de remplissage de dessin » à la page 405.

Vous trouverez ci-dessous le code ActionScript associé à cet exemple : Utilisez cette classe en tant que classe d'application principale d'un projet ActionScript uniquement dans Flex, ou en tant que classe de document du fichier FLA dans l'outil de programmation Flash :

```
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);

            shader.data.point1.value = [topMiddle.x, topMiddle,y];
            shader.data.point2.value = [bottomLeft.x, bottomLeft.y];
            shader.data.point3.value = [bottomRight.x, bottomRight.y];
            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();
        }
    }
}
```

Vous trouverez ci-dessous le code source du noyau du shader ColorFilter, utilisé pour la création du fichier « ColorFilter.pbj » de pseudo-code binaire Pixel Bender.

```
<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;
    }
}
```

Si vous utilisez un shader dont les paramètres ne sont pas documentés, c'est par le contrôle de la propriété `type` de l'objet ShaderParameter que vous pouvez évaluer le nombre d'éléments et leur type qu'il faut inclure dans le tableau. La propriété `type` indique le type de données du paramètre tel qu'il est défini dans le shader lui-même. Pour une liste du nombre et du type d'éléments prévus par chaque type de paramètre, consultez le répertoire des propriétés `ShaderParameter.value` dans le Guide de référence du langage.

Chaque objet ShaderParameter possède également une propriété `index` qui indique l'emplacement du paramètre dans l'ordre des paramètres du shader. En plus de ces propriétés, un objet ShaderParameter peut avoir des propriétés supplémentaires qui contiennent des valeurs de métadonnées fournies par le programmeur du shader. Par exemple, le programmeur peut définir pour un paramètre des valeurs de métadonnées telles que minimum, maximum et des valeurs par défaut. Toutes les valeurs de métadonnées spécifiées par le programmeur sont ajoutées à l'objet ShaderParameter en tant que propriétés dynamiques. Pour passer en revue ces propriétés, utilisez une boucle `for..in` pour boucler dans les propriétés dynamiques de l'objet ShaderParameter afin d'identifier ses métadonnées. L'exemple ci-dessous montre comment utiliser une boucle `for..in` afin d'identifier les métadonnées d'un objet ShaderParameter. Chaque valeur de métadonnées est ajoutée à une occurrence de Vector appelée `metadata`. Remarquez que cet exemple suppose qu'une occurrence de Shader appelée `myShader` existe déjà et qu'un paramètre appelé `brightness` également.

```
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
```

Utilisation d'un shader

Dès qu'un shader de Pixel Bender est disponible dans ActionScript en tant qu'objet Shader, il peut être utilisé de différentes façons :

- Remplissage d'un dessin par le shader : le shader spécifie quelle portion d'une forme dessinée utilise l'api du dessin
- Mode de fondu : le shader spécifie le degré de fondu entre deux objets d'affichage superposés
- Filtre : le shader définit un filtre qui modifie l'apparence du contenu visuel
- Traitement d'un shader autonome : le traitement du shader se fait sans définir l'usage de la sortie. Sur demande, le shader peut tourner en arrière-plan de telle sorte que le résultat devient disponible à la fin du traitement. Cette technique peut être utilisée pour générer des données bitmap et traiter des données non visuelles.

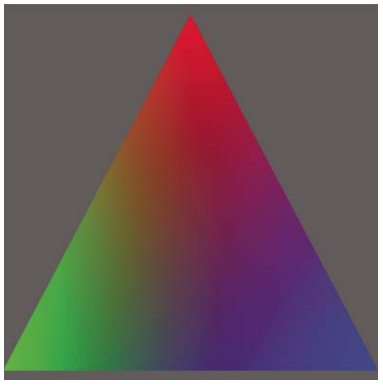
Utilisation d'un shader comme outil de remplissage de dessin

Si vous utilisez un shader pour créer un remplissage de dessin, vous faites appel aux méthodes de l'API de dessin pour créer une forme vectorielle. La sortie du shader permet de remplir la forme, à l'instar de toute image bitmap utilisée en tant que remplissage de bitmap par l'API de dessin. Pour créer un remplissage de dessin, appelez la méthode `beginShaderFill()` de l'objet Graphics à l'emplacement du code où vous souhaitez commencer à dessiner la forme. Transmettez l'objet Shader en tant que premier argument à la méthode `beginShaderFill()`, comme illustré dans le code suivant :

```
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
```

Lorsque vous utilisez un shader en tant qu'outil de remplissage de dessin, vous définissez les valeurs d'image d'entrée et de paramètre requises par le shader.

L'exemple suivant illustre l'utilisation d'un shader en tant qu'outil de remplissage de dessin. Dans cet exemple, le shader crée un dégradé à trois sommets. Ce dégradé possède trois couleurs, une par sommet d'un triangle, séparées des autres par un fondu dégradé. En outre, les couleurs pivotent pour créer un effet de rotation de couleur animé.



Remarque : *le code de cet exemple a été rédigé par Petri Leskinen. Merci Petri de bien vouloir nous en faire profiter. Vous pouvez consulter d'autres exemples et didacticiels rédigés par Petri à l'adresse http://pixelero.wordpress.com/.*

Le code ActionScript réside dans trois méthodes :

- `init()` : la méthode `init()` est appelée lors du chargement de l'application. Dans cette méthode, le code définit les valeurs initiales des objets Point qui représentent les sommets du triangle. Il crée également une occurrence de Sprite appelée `canvas`. Ultérieurement, dans la méthode `updateShaderFill()`, le code dessine une fois par image le résultat du shader dans `canvas`. Enfin, le code charge le fichier de pseudo-code binaire du shader.
- `onLoadComplete()` : dans la méthode `onLoadComplete()`, le code crée l'objet Shader appelé `shader`. Il définit également les valeurs initiales des paramètres. Enfin, le code ajoute la méthode `updateShaderFill()` en tant qu'écouteur de l'événement `enterFrame`, ce qui signifie qu'il est appelé une fois par image pour créer un effet animé.
- `onEnterFrame()` : la méthode `updateShaderFill()` est appelée une fois par image, ce qui crée l'effet d'animation. Dans cette méthode, le code calcule et définit les valeurs des paramètres du shader. Il appelle ensuite la méthode `beginShaderFill()` pour créer un remplissage de shader et appelle d'autres méthodes de l'API de dessin pour dessiner le résultat du shader dans un triangle.

Vous trouverez ci-dessous le code ActionScript associé à cet exemple : Utilisez cette classe en tant que classe d'application principale d'un projet ActionScript uniquement dans Flex, ou en tant que classe de document du fichier FLA dans l'outil de programmation Flash :

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

Le code source du noyau du shader ThreePointGradient, utilisé pour la création du fichier « ThreePointGradient.pbj » de pseudo-code binaire Pixel Bender, est indiqué ci-dessous :

```
<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);
    }
}
```

Pour plus d'informations sur le tracé de forme par le biais de l'API de dessin, consultez le chapitre « Utilisation de l'API de dessin » à la page 328.

Utilisation d'un shader comme mode de fondu

Utiliser un shader comme mode de fondu s'apparente à l'utilisation des autres modes de fondu. Le shader définit le résultat de deux objets d'affichage fondus visuellement. Pour utiliser un shader en tant que mode de fondu, affectez à votre objet Shader la propriété `blendShader` de l'objet d'affichage en premier plan. Affecter une valeur autre que `null` à la propriété `blendShader` définit automatiquement la propriété `blendMode` de l'objet d'affichage sur `BlendMode.SHADER`. Le code suivant illustre l'utilisation d'un shader en tant que mode de fondu. Notez que cet exemple part du principe qu'un objet d'affichage appelé `foreground` figure dans le même parent que l'autre contenu d'affichage dans la liste d'affichage, sachant que `foreground` chevauche l'autre contenu :

```
foreground.blendShader = myShader;
```

Lorsque vous utilisez un shader en tant que mode de fondu, deux entrées au moins doivent être spécifiées. Comme indiqué dans l'exemple, vous ne définissez pas les valeurs d'entrée dans le code. Les deux images fondues sont automatiquement utilisées en tant qu'entrées du shader. L'image en premier-plan fait office de seconde image. (C'est à cet objet d'affichage qu'est appliqué le mode de fondu.) Une image d'arrière-plan est créée à partir du composite de tous les pixels figurant derrière le cadre de délimitation de l'image en premier-plan. Cette image en arrière-plan est définie comme la première image d'entrée. Si vous utilisez un shader qui attend plus de deux entrées, vous en indiquez la valeur à partir de la troisième entrée.

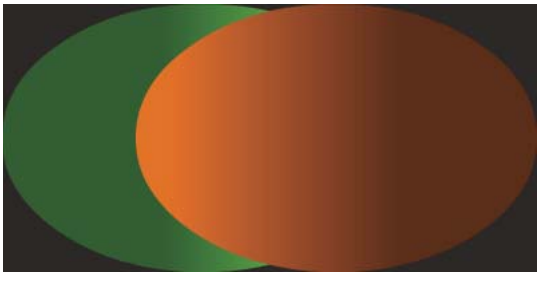
L'exemple suivant illustre l'utilisation d'un shader en tant que mode de fondu. Cet exemple utilise un mode de fondu Eclaircir basé sur la luminosité. Ce fondu a pour résultat d'afficher le pixel le plus clair de l'un ou l'autre des objets fondus.

Remarque : *le code de cet exemple a été rédigé par Mario Klingemann. Merci Mario de bien vouloir nous en faire profiter. Pour consulter d'autres exemples de code rédigés par Mario, ainsi que ses commentaires, visitez l'adresse www.quasimondo.com/.*

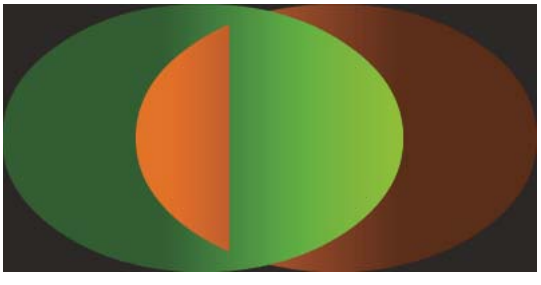
Le code ActionScript important figure dans les deux méthodes suivantes :

- `init()` : la méthode `init()` est appelée lors du chargement de l'application. Dans cette méthode, le code charge le fichier de pseudo-code binaire du shader.
- `onLoadComplete()` : dans la méthode `onLoadComplete()`, le code crée l'objet Shader appelé `shader`. Il dessine ensuite trois objets. Le premier, `backdrop`, est un arrière-plan gris foncé placé derrière les objets fondus. Le deuxième, `backgroundShape`, est une ellipse dégradée verte. Le troisième, `foregroundShape`, est une ellipse dégradée orange.

  L'ellipse `foregroundShape` est l'objet de premier plan du fondu. L'image d'arrière-plan du fondu est composée de la section de `backdrop` et de la section de `backgroundShape` qui sont recouvertes par le cadre de délimitation de l'objet `foregroundShape`. L'objet `foregroundShape` est affiché en première position dans la liste d'affichage. Il recouvre partiellement `backgroundShape` et totalement `backdrop`. En raison de ce chevauchement, sans application d'un mode de fondu, l'ellipse orange (`foregroundShape`) est entièrement affichée et masque une section de l'ellipse verte (`backgroundShape`) :



  Toutefois, si le mode de fondu est affiché, la section la plus lumineuse de l'ellipse verte est visible, car elle est plus claire que la section de `foregroundShape` qui la recouvre :



Vous trouverez ci-dessous le code ActionScript associé à cet exemple : Utilisez cette classe en tant que classe d'application principale d'un projet ActionScript uniquement dans Flex, ou en tant que classe de document du fichier FLA dans l'outil de programmation Flash :

```
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;
        }
    }
}
```

Le code source du noyau du shader LumaLighten, utilisé pour la création du fichier « LumaLighten.pbj » de pseudo-code binaire Pixel Bender, est indiqué ci-dessous :

```
<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;
    }
}
```

Pour plus d'informations sur l'utilisation des modes de fondu, consultez la section « Application de modes de fondu » à la page 311.

Utilisation d'un shader comme filtre

Utiliser un shader comme filtre s'apparente à l'utilisation de tout autre filtre dans ActionScript. Lorsque vous utilisez un shader en tant que filtre, l'image filtrée (un objet d'affichage ou un objet BitmapData) est transmise au shader. Le shader utilise l'image d'entrée pour créer le résultat du filtre, qui correspond généralement à une version modifiée de l'image d'origine. Si l'objet filtré est un objet d'affichage, le résultat du filtre remplace l'objet d'affichage filtré à l'écran. Si l'objet filtré est un objet BitmapData, le résultat du filtre devient le contenu de l'objet BitmapData dont la méthode `applyFilter()` est appelée.

Pour utiliser un shader en tant que filtre, vous commencez par créer l'objet Shader, comme indiqué à la section « Chargement ou intégration d'un shader » à la page 397. Vous créez ensuite un objet ShaderFilter lié à l'objet Shader. L'objet ShaderFilter est le filtre appliqué à l'objet filtré. Vous l'appliquez à un objet à l'instar de n'importe quel filtre. Vous le transmettez à la propriété `filters` d'un objet d'affichage ou vous appelez la méthode `applyFilter()` sur un objet BitmapData. Par exemple, le code suivant crée un objet ShaderFilter et applique le filtre à un objet d'affichage appelé `homeButton`.

```
var myFilter:ShaderFilter = new ShaderFilter(myShader);
homeButton.filters = [myFilter];
```

Lorsque vous utilisez un shader en tant que filtre, une entrée au moins doit lui être associée. Comme indiqué dans l'exemple, vous ne définissez pas la valeur d'entrée dans le code. L'objet d'affichage filtré ou l'objet BitmapData est défini en tant qu'image d'entrée. Si vous utilisez un shader qui attend plus d'une entrée, vous en spécifiez la valeur à partir de la deuxième entrée.

Dans certains cas, un filtre modifie les dimensions de l'image d'origine. Un effet d'ombre portée standard ajoute par exemple des pixels contenant l'ombre associée à l'image. Lorsque vous utilisez un shader qui modifie les dimensions de l'image, définissez les propriétés `leftExtension`, `rightExtension`, `topExtension` et `bottomExtension` pour indiquer l'écart approprié.

L'exemple suivant illustre l'utilisation d'un shader en tant que filtre. Dans cet exemple, le filtre inverse les valeurs des canaux rouge, vert et bleu d'une image. Il a pour résultat un « négatif » de l'image.

Remarque : *cet exemple utilise comme shader le noyau invertRGB.pbk Pixel Bender intégré à Pixel Bender Toolkit. Vous pouvez charger le code source du noyau à partir du répertoire d'installation de Pixel Bender Toolkit. Compilez le code source, puis enregistrez le fichier de pseudo-code binaire dans le même répertoire que le code source.*

Le code ActionScript important figure dans les deux méthodes suivantes :

- `init()` : la méthode `init()` est appelée lors du chargement de l'application. Dans cette méthode, le code charge le fichier de pseudo-code binaire du shader.
- `onLoadComplete()` : dans la méthode `onLoadComplete()`, le code crée l'objet Shader appelé `shader`. Il crée et dessine ensuite le contenu d'un objet appelé `target`. L'objet `target` est un rectangle rempli d'un dégradé linéaire rouge sur la gauche, jaune-vert au centre et bleu sur la droite. Si le filtre n'est pas appliqué, l'apparence de l'objet est la suivante :

Si le filtre est appliqué, les couleurs sont inversées, auquel cas l'apparence du rectangle est la suivante :

Ce code utilise à titre de shader le noyau « invertRGB.pbk » Pixel Bender d'exemple intégré à Pixel Bender Toolkit. Le code source figure dans le fichier « invertRGB.pbk », qui réside dans le répertoire d'installation de Pixel Bender Toolkit. Compilez le code source et enregistrez le fichier de pseudo-code binaire sous le nom « invertRGB.pbj » dans le même répertoire que votre code source ActionScript.

Vous trouverez ci-dessous le code ActionScript associé à cet exemple : Utilisez cette classe en tant que classe d'application principale d'un projet ActionScript uniquement dans Flex, ou en tant que classe de document du fichier FLA dans l'outil de programmation Flash :

```
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];
        }
    }
}
```

Pour plus d'informations sur l'application de filtres, consultez la section « Création et application de filtres » à la page 364.

Utilisation d'un shader en mode autonome

Lorsque vous utilisez un shader en mode autonome, son traitement s'exécute indépendamment de l'usage prévu du résultat. Vous spécifiez le shader à exécuter, définissez les valeurs d'entrée et de paramètres, puis stipulez un objet dans lequel seront placées les données résultantes. Vous pouvez utiliser un shader en mode autonome pour deux raisons :

- Traitement de données non graphiques : en mode autonome, vous pouvez choisir de transmettre des données binaires ou numériques arbitraires au shader au lieu de données d'image bitmap. Le cas échéant, le résultat du shader peut être renvoyé sous forme de données binaires ou numériques en plus des données d'image bitmap.
- Traitement en arrière-plan : lorsque vous exécutez un shader en mode autonome, il est exécuté en mode asynchrone par défaut. Cela signifie que le shader s'exécute en arrière-plan pendant que votre application continue à tourner et que votre code est averti une fois le traitement du shader terminé. Vous pouvez utiliser un shader sans pour autant bloquer l'interface utilisateur de l'application ou tout autre traitement, même si la durée de son exécution est élevée.

Un objet ShaderJob permet d'exécuter un shader en mode autonome. Vous commencez par créer un objet ShaderJob et vous le liez à l'objet Shader qui représente le shader à exécuter :

```
var job:ShaderJob = new ShaderJob(myShader);
```

Vous définissez ensuite toute valeur d'entrée ou de paramètre attendue par le shader. Si vous exécutez le shader en arrière-plan, vous enregistrez également un écouteur associé à l'événement `complete` de l'objet ShaderJob. Votre écouteur est appelé lorsque le shader termine sa tâche :

```
function completeHandler(event:ShaderEvent):void
{
    // do something with the shader result
}

job.addEventListener(ShaderEvent.COMPLETE, completeHandler);
```

Vous créez ensuite un objet dans lequel est écrit le résultat de l'opération du shader, une fois celle-ci terminée. Vous affectez cet objet à la propriété `target` de l'objet ShaderJob :

```
var jobResult:BitmapData = new BitmapData(100, 75);
job.target = jobResult;
```

Affectez une occurrence de BitmapData à la propriété `target` si vous utilisez l'objet ShaderJob pour exécuter le traitement de l'image. Si vous traitez des données binaires ou numériques, affectez un objet ByteArray ou une occurrence de Vector.<Number> à la propriété `target`. Si tel est le cas, vous devez définir les propriétés `width` et `height` de l'objet ShaderJob pour stipuler le volume de données à créer dans l'objet `target`.

Remarque : *vous pouvez définir les propriétés `shader`, `target`,`width` et `height` de l'objet ShaderJob en une seule étape. Pour ce faire, transmettez les arguments au constructeur `ShaderJob()`, comme suit :`var job:ShaderJob = new ShaderJob(myShader, myTarget, myWidth, myHeight);`.*

Lorsque vous êtes prêt à exécuter le shader, vous appelez la méthode `start()` de l'objet ShaderJob :

```
job.start();
```

Par défaut, appeler `start()` entraîne l'exécution asynchrone de l'objet ShaderJob. Dans ce cas, l'exécution du programme continue et passe immédiatement à la ligne suivante de code au lieu d'attendre que le shader soit terminé. Une fois l'opération du shader terminée, l'objet ShaderJob appelle ses écouteurs d'événement `complete` et les avertit. A ce stade (en d'autres termes, dans le corps de votre écouteur d'événement `complete`), l'objet `target` contient le résultat de l'opération du shader.

Remarque : *au lieu d'utiliser l'objet propriété `target`, vous pouvez extraire directement le résultat du shader de l'objet événement qui est transmis à la méthode d'écouteur. L'objet événement est une occurrence de ShaderEvent. L'objet ShaderEvent possède trois propriétés susceptibles d'être utilisées pour accéder au résultat, selon le type de données de l'objet défini en tant que propriété `target` : `ShaderEvent.bitmapData`, `ShaderEvent.byteArray` et `ShaderEvent.vector`.*

Vous pouvez également transmettre un argument `true` à la méthode `start()`. Dans ce cas, l'opération du shader s'exécute en mode synchrone. Tout le code (y compris l'interaction avec l'interface utilisateur et tout autre événement) est interrompu pendant l'exécution du shader. Une fois le shader terminé, l'objet `target` en contient le résultat et le programme passe à la ligne de code suivante.

```
job.start(true);
```

Chapitre 18 : Utilisation des clips

La classe MovieClip est la classe de base des animations et des symboles de clip créés dans Adobe® Flash® CS4 Professional. Elle possède tous les comportements et toutes les fonctionnalités des objets d'affichage, mais intègre des propriétés et méthodes supplémentaires qui permettent de contrôler le scénario d'un clip. Ce chapitre explique comment utiliser ActionScript pour gérer la lecture d'un clip et comment créer un clip de manière dynamique.

Principes de base des clips

Introduction à l'utilisation des clips

Les clips sont un élément capital pour les créateurs de contenu animé dans l'environnement de programmation Flash et pour contrôler ce contenu avec ActionScript. Lorsque vous créez un symbole de clip dans Flash, le programme ajoute ce symbole à la bibliothèque du document Flash. Par défaut, ce symbole devient une occurrence de la classe MovieClip et, de ce fait, possède les propriétés et méthodes de la classe MovieClip.

Lorsqu'une occurrence d'un symbole de clip est placée sur la scène, la progression du scénario de ce clip s'effectue automatiquement (si le clip comporte plusieurs images), sauf si cette lecture est modifiée en ActionScript. Le scénario est l'élément distinctif de la classe MovieClip. Il permet de créer des animations par interpolation de mouvement ou de forme via l'interface de Flash. Par contre, un objet d'affichage issu de la classe Sprite peut uniquement être animé par programmation, en modifiant ses valeurs.

Dans les versions précédentes d'ActionScript, la classe MovieClip constituait la classe de base de toutes les occurrences de la scène. En ActionScript 3.0, un clip est désormais un objet d'affichage parmi d'autres, tous susceptibles de s'afficher à l'écran. S'il n'est pas nécessaire de définir un scénario pour la fonction d'un objet d'affichage, l'utilisation de la classe Shape au lieu de la classe MovieClip peut accroître les performances d'affichage. Pour plus d'informations sur le choix des objets d'affichage en fonction de la tâche prévue, consultez la section « Sélection d'une sous-classe de DisplayObject » à la page 298.

Tâches courantes d'utilisation des clips

Voici quelques tâches courantes relatives aux clips qui sont présentées dans ce chapitre :

- Lecture et arrêt des clips
- Lecture des clips en marche arrière.
- Déplacement de la tête de lecture à un point spécifique du scénario d'un clip
- Utilisation des étiquettes d'images en ActionScript
- Accès aux informations de séquence en ActionScript
- Création d'occurrences de symboles de clips dans la bibliothèque en ActionScript
- Chargement et contrôle de fichiers SWF externes, y compris s'ils ont été créés avec des versions antérieures de Flash Player
- Construction d'un système en ActionScript pour la création d'éléments graphiques à charger lors de l'exécution

Concepts importants et terminologie

La liste de référence suivante contient les termes importants utilisés dans ce chapitre :

- SWF AVM1 : fichier SWF créé en ActionScript 1.0 ou ActionScript 2.0, et généralement destiné aux versions 8 ou antérieures de Flash Player.
- SWF AVM2 : fichier SWF créé en ActionScript 3.0 pour Adobe Flash Player 9 ou ultérieur ou Adobe AIR.
- SWF externe : fichier SWF créé séparément de celui du projet, et destiné à être chargé et lu dans celui-ci.
- Image : élément de base du scénario. Comme les images de films, chaque image est une « photographie » et c'est la lecture rapide des images en séquence qui produit l'effet d'animation.
- Scénario : métaphore représentant la série d'images qui composent la séquence d'animation d'un clip. Le scénario d'un objet MovieClip est l'équivalent du scénario de cet objet dans l'environnement de programmation Flash.
- Tête de lecture : marqueur identifiant la position (l'image) dans le scénario qui est affichée à un moment donné.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester des exemples de code. Etant donné que ce chapitre traite de l'utilisation de clips dans ActionScript, les exemples de code proposés sont principalement écrits dans l'idée de manipuler un symbole de clip ayant été créé et placé sur la Scène. Pour tester un exemple, il est nécessaire d'afficher le résultat dans Flash Player ou AIR afin de voir l'effet du code sur le symbole. Pour tester les codes de ce chapitre :

1. Créez un document Flash vide.
2. Sélectionnez une image-clé dans le scénario.
3. Ouvrez le panneau Actions et copiez le code dans le panneau Script.
4. Créez une occurrence de symbole de clip sur la scène. Par exemple, dessinez une forme, sélectionnez-la et choisissez Modification > Convertir en symbole. Donnez ensuite un nom au symbole.
5. Sélectionnez le clip, puis donnez-lui un nom d'occurrence dans l'Inspecteur des Propriétés. Le nom doit correspondre au nom utilisé pour le clip dans l'exemple de code (par exemple, si le code manipule un clip `myMovieClip`, vous devez appeler votre occurrence de clip `myMovieClip` également).
6. Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

   Les résultats du code manipulant le clip s'affichent à l'écran, comme indiqué dans le code.

D'autres techniques de test d'exemples de code sont expliquées plus en détail à la section « Test des exemples de code contenus dans un chapitre » à la page 36.

Utilisation des objets MovieClip

Lorsque vous publiez un fichier SWF, Flash convertit toutes les occurrences de symboles de clips sur la scène en objets MovieClip. Pour qu'un symbole de clip soit disponible en ActionScript, donnez-lui un nom d'occurrence dans le champ Nom de l'occurrence de l'Inspecteur des Propriétés. Lors de la création du fichier SWF, Flash génère le code qui crée l'occurrence de MovieClip sur la scène et déclare une variable avec ce nom d'occurrence. Si les clips que vous avez nommés sont imbriqués dans d'autres clips nommés, ces clips enfant sont traités comme des propriétés du clip parent : vous pouvez accéder à tous les clips de cette hiérarchie en utilisant la syntaxe à point. Par exemple, si une occurrence de clip appelée `childClip` est imbriquée dans une autre appelée `parentClip`, vous pouvez lancer sa lecture en appelant le code suivant :

```
parentClip.childClip.play();
```

Remarque : *les occurrences enfant placées sur la scène dans l'outil de programmation de Flash ne sont pas accessibles par le code depuis l'intérieur du constructeur d'une occurrence parent car, à ce stade de l'exécution du code, elles n'ont pas été créées. Avant d'accéder à l'enfant, le parent doit soit créer l'occurrence enfant par du code, soit retarder l'accès à une fonction de rappel qui écoute l'enfant en vue de la distribution de son événement* `Event.ADDED_TO_STAGE`*.*

Si de nombreuses méthodes et propriétés de la classe MovieClip d'ActionScript 2.0 restent les mêmes, d'autres ont changé. Toutes les propriétés qui étaient préfixées d'un trait de soulignement ont été renommées. Par exemple, les propriétés `_width` et `_height` sont désormais accessibles sous le nom `width` et `height`, `_xscale` et `_yscale` sous le nom`scaleX` et `scaleY`. Pour obtenir la liste complète des propriétés et méthodes de la classe MovieClip, reportez-vous au Guide de référence du langage et des composants ActionScript 3.0.

Contrôle de la lecture d'un clip

Flash utilise la métaphore du scénario pour traduire une animation ou un changement d'état. Tout élément visuel qui a recours à un scénario est soit un objet MovieClip, soit une extension de la classe MovieClip. Bien qu'il soit possible en ActionScript de commander l'arrêt ou la lecture d'un clip et le passage à un autre point du scénario, il n'est pas possible de créer dynamiquement un scénario ni d'ajouter du contenu dans une image spécifique. Seul l'outil de programmation Flash le permet.

Pendant la lecture, le clip progresse le long du scénario à une vitesse fixée par la cadence d'image du fichier SWF. Vous pouvez également ignorer ce paramètre et le remplacer par la propriété `Stage.frameRate` dans ActionScript.

Lecture et arrêt des clips

Les méthodes `play()` et `stop()` permettent d'effectuer des manipulations simples sur un clip tout au long du scénario. Par exemple, supposons qu'un symbole de clip sur la scène contienne une animation représentant une bicyclette qui traverse l'écran, et dont le nom d'occurrence est `bicycle`. Si le code suivant est associé à une image-clé du scénario principal,

```
bicycle.stop();
```

la bicyclette ne se déplace pas (son animation n'est pas mise en lecture). Le mouvement de la bicyclette peut être déclenché par une action de l'utilisateur. Par exemple, avec un bouton nommé `startButton`, le code suivant (dans une image-clé du scénario principal) déclenche l'animation si l'utilisateur clique sur ce bouton :

```
// 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);
```

Avance rapide et rembobinage

Les méthodes `play()` et `stop()` ne sont pas les seules méthodes commandant l'animation d'un clip. Vous pouvez également avancer et reculer manuellement la tête de lecture à l'aide des méthodes `nextFrame()` et `prevFrame()`. L'appel de l'une de ces deux méthodes arrête la lecture et fait avancer le clip ou le rembobine d'une image.

L'utilisation de la méthode `play()` revient à appeler `nextFrame()` chaque fois qu'un événement `enterFrame` est déclenché pour cet objet clip. Vous pouvez donc envisager de lire le clip `bicycle` en arrière en ajoutant un écouteur pour l'événement `enterFrame` et en indiquant à `bicycle`, dans la fonction écouteur, de reculer d'une image, comme suit :

```
// 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);
```

En lecture normale, si un clip contient plusieurs images, il tourne en boucle lors de la lecture, c'est-à-dire qu'il revient à l'image 1 une fois qu'il a passé la dernière image. Si vous utilisez `prevFrame()` ou `nextFrame()`, ce comportement ne se produit pas automatiquement (l'appel de `prevFrame()` lorsque la tête de lecture est sur l'image 1 ne déplace pas la tête de lecture à la dernière image). Dans l'exemple ci-dessus, la condition `if` vérifie si le clip est revenu à la première image et fait alors passer le clip à la dernière image, créant ainsi une boucle continue de lecture en arrière.

Déplacement vers une autre image et utilisation des étiquettes d'image

Le déplacement d'un clip à une nouvelle image est une opération simple. Il suffit d'appeler `gotoAndPlay()` ou `gotoAndStop()` en spécifiant le numéro de l'image cible comme paramètre. Vous pouvez également transmettre une chaîne correspondant au nom de l'étiquette d'image. Il est possible d'attribuer une étiquette à toute image du scénario. Pour ce faire, sélectionnez une image du scénario, puis tapez un nom dans le champ Etiquette d'image de l'Inspecteur des Propriétés.

L'utilisation des étiquettes d'image plutôt que des numéros présente des avantages évidents pour créer un clip complexe. Si le nombre d'images, de calques et d'interpolations d'une animation est élevé, envisagez d'étiqueter les images importantes de manière évocatrice, en faisant référence aux changements de comportement dans le clip (par exemple « départ », « marche » ou « course »). Cette technique permet d'améliorer la lisibilité du code et offre plus de souplesse, puisque les appels ActionScript destinés à une image étiquetée pointent sur une référence uniquement (l'étiquette) plutôt que sur une image numérotée. Si vous décidez par la suite de déplacer un segment de l'animation vers une autre image, il ne sera pas nécessaire de modifier le code ActionScript si vous conservez les mêmes étiquettes pour toutes les images au nouvel emplacement.

Pour représenter des étiquettes en code, ActionScript 3.0 comporte la classe FrameLabel. Chaque occurrence de cette classe représente une étiquette d'image unique, et possède une propriété `name` qui représente le nom de l'étiquette tel qu'il a été indiqué dans l'Inspecteur des Propriétés, ainsi qu'une propriété `frame` qui représente le numéro de l'image pour laquelle l'étiquette est placée dans le scénario.

Pour permettre d'accéder aux occurrences de FrameLabel associées à une occurrence de clip, la classe MovieClip comporte deux propriétés qui renvoient directement des objets FrameLabel. La propriété `currentLabels` renvoie un tableau composé de tous les objets FrameLabel présents sur l'ensemble du scénario d'un clip. La propriété `currentLabel` renvoie une chaîne contenant le nom de l'étiquette d'image trouvée récemment sur le scénario.

Supposons que vous ayez créé un clip nommé `Robot` et que vous ayez étiqueté les différents états de l'animation. Vous pouvez définir une condition pour vérifier la propriété `currentLabel` afin d'accéder à l'état actuel de `Robot`, comme dans le code suivant :

```
if (robot.currentLabel == "walking")
{
    // do something
}
```

Utilisation des séquences

Dans l'environnement de programmation Flash, vous pouvez utiliser les séquences pour démarquer une série de scénarios qu'un fichier SWF doit suivre. Grâce au second paramètre de la méthode `gotoAndPlay()` ou `gotoAndStop()`, vous pouvez spécifier la séquence sur laquelle la tête de lecture doit se placer. Tous les fichiers FLA commencent par une séquence, à laquelle vous pouvez ajouter le nombre de séquences de votre choix.

L'utilisation des séquences n'est pas toujours conseillée, car elle présente un certain nombre d'inconvénients. La maintenance d'un document Flash qui contient de nombreuses séquences peut être difficile, en particulier dans les environnements où plusieurs auteurs interviennent. Un nombre élevé de séquences peut également s'avérer inefficace de point de vue de la bande passante, car le processus de publication implique la fusion de toutes les séquences en un seul scénario. L'ensemble des séquences est alors téléchargé en mode progressif, même si elles ne sont jamais lues. C'est pourquoi l'utilisation de plusieurs séquences est largement déconseillée, à moins qu'elle soit nécessaire à l'organisation de plusieurs animations basées sur des scénarios.

La propriété `scenes` de la classe MovieClip renvoie un tableau des objets Scene représentant toutes les séquences du fichier SWF. La propriété `currentScene` renvoie un objet Scene représentant la séquence qui est en cours de lecture.

La classe Scene a des propriétés qui contiennent des informations sur la séquence. La propriété `labels` renvoie un tableau des objets FrameLabel utilisés au sein de la séquence. La propriété `name` renvoie le nom de la séquence sous forme de chaîne. La propriété `numFrames` renvoie un entier représentant le nombre total d'images dans la séquence.

Création d'objets MovieClip à l'aide d'ActionScript

L'ajout de contenu s'effectue dans Flash en faisant glisser des éléments de la bibliothèque sur la scène. Mais il ne s'agit pas là de la seule méthode. Pour des projets plus complexes, les développeurs expérimentés préfèrent souvent créer les clips par programmation. Cette approche présente plusieurs avantages: réutilisation facile du code, vitesse de compilation accrue et disponibilité de modifications plus sophistiquées réservées à ActionScript.

La nouvelle API de liste d'affichage d'ActionScript 3.0 simplifie le processus de création dynamique d'objets MovieClip. La possibilité d'instancier directement une occurrence de MovieClip, séparément de son ajout à la liste d'affichage, offre beaucoup de souplesse et de simplicité sans sacrifier tout contrôle.

En ActionScript 3.0, lorsqu'une occurrence de clip (ou de tout autre objet d'affichage) est créée par code, elle est invisible tant qu'elle n'a pas été ajoutée à la liste d'affichage à l'aide de la méthode `addChild()` ou `addChildAt()` d'un conteneur d'objets d'affichage Vous pouvez ainsi créer un clip et définir ses propriétés, et même appeler ses méthodes, avant qu'il apparaisse à l'écran. Pour plus d'informations sur l'utilisation de la liste d'affichage, consultez la section « Utilisation de conteneurs d'objets d'affichage » à la page 286.

Exportation des symboles de bibliothèque pour ActionScript

Par défaut, il est impossible de créer dynamiquement des occurrences de symboles de clips dans la bibliothèque d'un document Flash (c'est-à-dire de les créer entièrement en ActionScript). En effet, l'exportation de chaque symbole pour une utilisation en ActionScript accroît la taille du fichier SWF. Or, les symboles ne sont pas tous destinés à une utilisation sur la scène. C'est pourquoi, pour qu'un symbole soit disponible en ActionScript, vous devez indiquer spécifiquement que ce symbole doit être exporté pour ActionScript.

Pour exporter un symbole pour ActionScript :

1. Sélectionnez le symbole dans le panneau Bibliothèque et ouvrez sa boîte de dialogue Propriétés.
2. Si nécessaire, activez les paramètres avancés.
3. Dans la boîte de dialogue Liaison, activez l'option Exporter pour ActionScript.

   Les champs Classe et Classe de base sont alors activés.

   Par défaut, le champ Classe contient le nom du symbole sans espaces (par exemple, un symbole nommé « Maison Rouge » devient « MaisonRouge »). Pour spécifier que le symbole doit utiliser une classe personnalisée pour son comportement, saisissez le nom complet de cette classe, package compris. Si vous voulez avoir la possibilité de créer des occurrences du symbole en ActionScript, sans avoir pour autant besoin de comportements supplémentaires, vous pouvez laisser ce nom de classe tel quel.

   Par défaut, le champ Classe de base a la valeur `flash.display.MovieClip`. Si vous voulez que votre symbole étende les fonctionnalités d'une autre classe personnalisée, vous pouvez remplacer cette valeur par le nom de votre classe, sous réserve que celle-ci étende la classe Sprite (ou MovieClip).
4. Cliquez sur OK pour enregistrer les changements.

   A ce stade, si Flash ne détecte pas de fichier ActionScript externe contenant la définition de la classe spécifiée (par exemple, si vous n'avez pas besoin d'ajouter un comportement supplémentaire pour le symbole), un message d'avertissement apparaît :

   Le chemin de classe ne contient pas de définition pour cette classe. Une définition sera générée automatiquement dans le fichier SWF lors de l'exportation.

   Vous pouvez ignorer cet avertissement si le symbole de bibliothèque ne nécessite pas de fonctionnalités en dehors de celles de la classe MovieClip.

Si vous n'indiquez aucune classe pour votre symbole, Flash crée pour lui une classe du type de celle-ci :

```
package
{
    import flash.display.MovieClip;

    public class ExampleMovieClip extends MovieClip
    {
        public function ExampleMovieClip()
        {
        }
    }
}
```

Si vous ne souhaitez pas ajouter des fonctionnalités ActionScript au symbole, ajoutez les propriétés et méthodes nécessaires à la structure de code suivante. Supposons par exemple que vous disposez d'un symbole de clip contenant un cercle de 50 pixels de diamètre, dont le paramètre de liaison est Exporter pour ActionScript et dont la classe est Circle. Le code suivant, lorsqu'il est placé dans un fichier Circle.as, étend la classe MovieClip et fournit au symbole les méthodes supplémentaires `getArea()` et `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;
        }
    }
}
```

Le code suivant, placé dans une image-clé à l'image 1 du document Flash, crée une occurrence du symbole et l'affiche à l'écran :

```
var c:Circle = new Circle();
addChild(c);
trace(c.width);
trace(c.height);
trace(c.getArea());
trace(c.getCircumference());
```

Ce code montre comment utiliser l'instanciation ActionScript au lieu de faire glisser des actifs individuels vers la scène. Il crée un cercle qui présente toutes les propriétés d'un clip et possède les méthodes personnalisées définies dans votre classe Circle. Il s'agit d'un exemple tout à fait élémentaire ; un symbole de bibliothèque peut spécifier un nombre illimité de propriétés et de méthodes dans sa classe.

L'instanciation en ActionScript est un processus puissant, car il permet de créer dynamiquement de nombreuses occurrences qu'il serait particulièrement laborieux de créer manuellement. Elle vous apporte en outre une grande souplesse, puisque vous pouvez personnaliser les propriétés de chaque occurrence dès sa création. Pour saisir l'importance de ces avantages, vous pouvez utiliser une boucle pour créer dynamiquement plusieurs occurrences de Circle. Après avoir ajouté le symbole Circle et la classe correspondante, décrits précédemment, dans la bibliothèque de votre document Flash, placez le code suivant dans une image-clé à l'image 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);
}
```

Cet exemple illustre la rapidité avec laquelle vous pouvez créer et personnaliser plusieurs occurrences d'un symbole à l'aide de code. Chaque occurrence est positionnée en fonction du compteur de boucle. Ensuite une couleur lui est attribuée au hasard à l'aide de la propriété `transform` (dont Circle hérite en étendant la classe MovieClip).

Chargement d'un fichier SWF externe

En ActionScript 3.0, les fichiers SWF sont chargés avec la classe Loader. Pour charger un fichier SWF externe, vous devez définir quatre étapes en ActionScript :

1. Créer un objet URLRequest avec l'adresse URL du fichier.
2. Créer un objet Loader.
3. Appeler la méthode `load()` de l'objet Loader en lui passant en paramètre l'occurrence de l'objet URLRequest.
4. Appeler la méthode `addChild()` pour un conteneur d'objet d'affichage (par exemple le scénario principal d'un document Flash) afin d'ajouter l'occurrence de Loader à la liste d'affichage

Le code final se présente ainsi :

```
var request:URLRequest = new URLRequest("http://www.[yourdomain].com/externalSwf.swf");
var loader:Loader = new Loader()
loader.load(request);
addChild(loader);
```

Le même code peut être utilisé pour charger un fichier image externe (image JPEG, GIF ou PNG) en spécifiant l'adresse URL de ce fichier à la place de celle d'un fichier SWF. Contrairement à un fichier image, un fichier SWF peut contenir du code ActionScript. Ainsi, bien que le processus de chargement d'un fichier SWF soit identique à celui d'une image, le fichier SWF à l'origine du chargement et le fichier SWF chargé doivent se trouver dans le même sandbox de sécurité si vous souhaitez utiliser ActionScript pour communiquer avec le fichier SWF externe. En outre, si ce fichier externe contient des classes qui partagent le même espace de noms que les classes du fichier SWF qui le charge, il peut s'avérer nécessaire de créer un domaine d'application pour le fichier chargé afin d'éviter d'éventuels conflits d'espace de nom. Pour plus d'informations sur la sécurité et le domaine d'application, consultez les sections « Utilisation de la classe ApplicationDomain » à la page 666 et « Chargement de fichiers SWF et d'images » à la page 728.

Une fois que le fichier SWF externe est chargé, il devient accessible via la propriété `Loader.content`. Si ce fichier est publié en ActionScript 3.0, il s'agira soit d'un clip, soit d'un sprite, selon la classe qu'il étend.

Considérations relatives au chargement de fichiers SWF de version ancienne

Si un fichier SWF externe a été publié dans une version antérieure d'ActionScript, il faut tenir compte de restrictions importantes. Contrairement à un fichier SWF ActionScript 3.0 qui s'exécute dans AVM2 (ActionScript Virtual Machine 2), un fichier SWF publié en ActionScript 1.0 ou 2.0 s'exécute dans AVM1 (ActionScript Virtual Machine 1).

Lorsque le chargement d'un fichier SWF AVM1 réussit, l'objet chargé (la propriété `Loader.content`) sera un objet AVM1Movie. Une occurrence d'AVM1Movie n'est pas identique à une occurrence de MovieClip. C'est un objet d'affichage qui, contrairement à un clip, ne comporte pas de méthodes ni de propriétés liées au scénario. Le fichier SWF AVM2 parent ne pourra pas accéder à ses propriétés, méthodes ou objets.

D'autres restrictions s'appliquent à un fichier SWF AVM1 chargé par un fichier SWF AVM2 : Pour plus de détails, reportez-vous à la description de la classe AVM1Movie dans le Guide de référence du langage et des composants ActionScript 3.0.

Exemple : RuntimeAssetsExplorer

La fonctionnalité Exporter pour ActionScript présente des avantages particuliers lorsque des bibliothèques peuvent être réutilisées sur plusieurs projets. Si Flash Player ou AIR exécute un fichier SWF, les symboles ayant été exportés dans ActionScript sont disponibles pour tous les fichiers SWF au sein du même sandbox de sécurité que le fichier SWF qui le charge. De cette manière, un seul document Flash peut générer un fichier SWF destiné uniquement à accueillir des éléments graphiques. Cette technique est très utile pour les grands projets dans lesquels les concepteurs graphiques chargés des éléments visuels peuvent travailler en parallèle avec les développeurs qui créent une « enveloppe » SWF qui chargera les fichiers SWF des éléments graphiques au moment de l'exécution. Cette méthode peut vous servir dans la maintenance d'un ensemble de versions de fichiers dans lesquelles les actifs graphiques ne sont pas dépendants du développement de la programmation.

L'application RuntimeAssetsExplorer charge tout fichier SWF qui est une sous-classe de RuntimeAsset et permet de consulter les éléments disponibles de ce fichier SWF. Cet exemple illustre les points suivants :

- Chargement d'un fichier SWF externe à l'aide de `Loader.load()`
- Création dynamique d'un symbole de bibliothèque exporté pour ActionScript
- Contrôle de la lecture du MovieClip avec ActionScript

Avant de commencer, notez que tous les fichiers SWF devant s'exécuter dans Flash Player doivent se trouver dans le même sandbox de sécurité. Pour plus d'informations, consultez la section « Les sandbox de sécurité » à la page 716.

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers de l'application RuntimeAssetsExplorer se trouvent dans le dossier Samples/Chapters/RuntimeAssetsExplorer. L'application se compose des fichiers suivants :

Fichier	Description
RuntimeAssetsExample.mxml ou RuntimeAssetsExample.fla	Interface utilisateur de l'application pour Flex (MXML) ou Flash (FLA).
RuntimeAssetsExample.as	Classe document pour l'application Flash (FLA).
GeometricAssets.as	Classe exemple qui implémente l'interface RuntimeAsset.
GeometricAssets.fla	Fichier FLA lié à la classe GeometricAssets (classe de document du fichier FLA) contenant les symboles exportés pour ActionScript.
com/example/programmingas3/runtimeassetexplorer/RuntimeLibrary.as	Interface qui définit les méthodes attendues par tous les fichiers SWF d'actifs d'exécution qui seront chargés dans l'explorateur.
com/example/programmingas3/runtimeassetexplorer/AnimatingBox.as	Classe du symbole de bibliothèque dont la forme est un rectangle pivotant.
com/example/programmingas3/runtimeassetexplorer/AnimatingStar.as	Classe du symbole de bibliothèque dont la forme est une étoile pivotante.

Etablissement de l'interface de bibliothèque d'exécution

Pour que l'explorateur interagisse convenablement avec une bibliothèque SWF, la structure des bibliothèques d'éléments doit être formalisée. Pour ce faire, nous allons créer une interface, que l'on pourrait comparer à une classe dans la mesure où elle représente un modèle de définition des méthodes qui définissent une structure attendue. Par contre, à l'inverse d'une classe, elle ne comporte pas les corps des méthodes. L'interface est un moyen de communication pour la bibliothèque d'éléments et l'explorateur. Chaque fichier SWF d'éléments chargé dans le navigateur implémentera cette interface. Pour plus d'informations sur l'utilité des interfaces, consultez la section « Interfaces » à la page 109.

L'interface RuntimeLibrary est très simple, puisque nous avons uniquement besoin d'une fonction qui fournisse à l'explorateur un tableau de chemins de classe pour les symboles à exporter disponibles dans la bibliothèque. Cette interface utilise pour ce faire une seule méthode : `getAssets()`.

```
package com.example.programmingas3.runtimeassetexplorer
{
    public interface RuntimeLibrary
    {
        function getAssets():Array;
    }
}
```

Création de fichiers SWF de bibliothèque d'actifs

La définition de l'interface RuntimeLibrary permet de créer plusieurs fichiers SWF de bibliothèque d'actifs, qui pourront être chargés dans un autre fichier SWF. L'élaboration d'une bibliothèque SWF d'actifs repose sur quatre tâches :

- Création d'une classe pour le fichier SWF de bibliothèque d'actifs

- Création de classes pour les différents éléments contenus dans la bibliothèque
- Création des éléments graphiques eux-mêmes
- Association des éléments graphiques à des classes et publication du fichier SWF de bibliothèque

Création d'une classe pour implémenter l'interface RuntimeLibrary

Nous allons ensuite créer la classe GeometricAssets qui implémente l'interface RuntimeLibrary, et sera la classe du document FLA. Le code de cette classe est très proche de celui de l'interface RuntimeLibrary, la différence étant que la définition de classe comporte le corps de la méthode `getAssets()`.

```
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" ];
        }
    }
}
```

Si nous devions créer une deuxième bibliothèque d'exécution, nous pourrions créer un autre fichier FLA reposant sur une autre classe (AnimationAssets, par exemple) qui assure sa propre implémentation de `getAssets()`.

Création de classes pour chaque élément MovieClip

Pour les besoins de cet exemple, nous étendrons simplement la classe MovieClip, sans ajouter de fonctionnalité aux éléments. Le code suivant destiné à AnimatingStar est semblable à celui de AnimatingBox :

```
package com.example.programmingas3.runtimeassetexplorer
{
    import flash.display.MovieClip;

    public class AnimatingStar extends MovieClip
    {
        public function AnimatingStar() {
        }
    }
}
```

Publication de la bibliothèque

Nous allons maintenant relier les actifs de classe MovieClip à la nouvelle classe en créant un fichier FLA et en entrant GeometricAssets dans le champ Classe du document de l'Inspecteur des propriétés. Pour les besoins de cet exemple, nous allons créer deux formes très simples qui utilisent une interpolation de scénario pour effectuer une rotation horaire sur 360 images. Les deux symboles `animatingBox` et `animatingStar` sont définis pour l'exportation vers ActionScript et leurs champs Classe correspondent aux chemins de classe respectifs spécifiés dans l'implémentation de `getAssets()`. La classe de base par défaut `flash.display.MovieClip` est conservée, puisque nous voulons créer des sous-classes pour les méthodes MovieClip standard.

Après définition des paramètres d'exportation de votre symbole, publiez le fichier FLA. Vous disposez maintenant de votre première bibliothèque d'exécution. Ce fichier SWF pourrait être chargé dans un autre fichier SWF AVM2, dans lequel les symboles AnimatingBox et AnimatingStar seraient disponibles.

Chargement de la bibliothèque dans un autre fichier SWF

Le dernier élément fonctionnel à traiter est l'interface utilisateur de l'explorateur. Dans cet exemple, le chemin d'accès à la bibliothèque d'exécution est codé en dur dans la variable `ASSETS_PATH`. Vous pourriez également utiliser la classe FileReference, par exemple pour créer une interface qui recherche un fichier SWF particulier sur le disque dur.

Une fois que la bibliothèque d'exécution est chargée, Flash Player appelle la méthode `runtimeAssetsLoadComplete()`.

```
private function runtimeAssetsLoadComplete(event:Event):void
{
    var rl:* = event.target.content;
    var assetList:Array = rl.getAssets();
    populateDropdown(assetList);
    stage.frameRate = 60;
}
```

Dans cette méthode, la variable rl représente le fichier SWF chargé. Le code appelle la méthode `getAssets()` du fichier SWF chargé, obtient la liste des éléments disponibles et remplit un composant ComboBox avec cette liste en appelant la méthode `populateDropDown()`. Cette méthode enregistre le chemin de classe complet de chaque élément. Si l'utilisateur clique sur le bouton Ajouter, l'interface déclenche la méthode `addAsset()` :

```
private function addAsset():void
{
    var className:String = assetNameCbo.selectedItem.data;
    var AssetClass:Class = getDefinitionByName(className) as Class;
    var mc:MovieClip = new AssetClass();
    ...
}
```

qui obtient le chemin de classe de l'élément actuellement sélectionné dans la ComboBox (`assetNameCbo.selectedItem.data`), et utilise la fonction `getDefinitionByName()` (du package flash.utils) pour obtenir une référence à la classe de l'élément afin de créer une nouvelle occurrence de celui-ci.

Chapitre 19 : Utilisation des interpolations de mouvement

La section « Animation des objets » à la page 317 décrit la procédure d'implémentation d'une animation pilotée par un script ActionScript.

Ce chapitre est consacré à une autre technique de création d'une animation : l'interpolation de mouvement. Cette technique vous permet de créer un mouvement en le configurant en mode interactif dans un fichier FLA par le biais d'Adobe® Flash® CS4 Professional. Vous pouvez ensuite utiliser ce mouvement dans votre animation dynamique à base de code ActionScript lors de l'exécution.

Flash CS4 génère automatiquement le code ActionScript qui implémente l'interpolation de mouvement et vous permet de le copier et de le réutiliser.

Pour créer des interpolations de mouvement, vous devez disposer d'une licence pour Adobe Flash CS4 Professional.

Principes de base des interpolations de mouvement

Introduction aux interpolations de mouvement dans ActionScript

Les interpolations de mouvement permettent de créer aisément une animation.

Une interpolation de mouvement modifie les propriétés d'un objet d'affichage, telles que la position ou la rotation, d'une image à l'autre. Elle permet également de modifier l'apparence d'un objet d'affichage en cours de déplacement en lui appliquant divers filtres et propriétés. Vous créez l'interpolation de mouvement en mode interactif dans Flash, ce qui génère le code ActionScript correspondant. Dans Flash, utilisez la commande Copie de mouvement en tant qu'ActionScript 3.0 pour copier le code ActionScript à l'origine de l'interpolation de mouvement. Vous pouvez alors réutiliser le code ActionScript pour créer un mouvement dans votre animation dynamique lors de l'exécution.

Pour plus d'informations sur la création d'une interpolation de mouvement, consultez la section Interpolations de mouvement du manuel *Utilisation de Flash CS4 Professional*.

Tâches d'interpolation de mouvement courantes

Le code ActionScript généré automatiquement qui implémente une interpolation de mouvement effectue les tâches suivantes :

- Il instancie une occurrence d'objet de mouvement associé à l'interpolation.
- Il définit la durée de l'interpolation de mouvement.
- Il ajoute les propriétés de l'interpolation de mouvement.
- Il ajoute des filtres à l'interpolation de mouvement.
- Il associe l'interpolation de mouvement aux objets d'affichage associés.

Terminologie et concepts importants

Un terme important que vous rencontrerez dans ce chapitre est défini ci-dessous :

- Interpolation de mouvement : construction qui génère des images intermédiaires d'un objet d'affichage dans des états et à des moments différents, créant ainsi l'impression que le premier état évolue progressivement vers le second état. Une interpolation de mouvement permet de déplacer un objet d'affichage sur la scène, ainsi que de le faire progressivement grandir, rétrécir, pivoter ou changer de couleur.

Copie de scripts d'interpolation de mouvement

Une interpolation génère des images intermédiaires qui affichent un objet d'affichage dans des états différents dans deux images distinctes d'un scénario. Elle crée l'impression que le contenu de la première image se transforme progressivement en contenu de la seconde image. Dans une interpolation de mouvement, le changement d'apparence implique généralement la modification de la position de l'objet d'affichage, créant ainsi un mouvement. Outre le repositionnement de l'objet d'affichage, une interpolation de mouvement peut faire pivoter, incliner ou redimensionner ce dernier, voire lui appliquer des filtres.

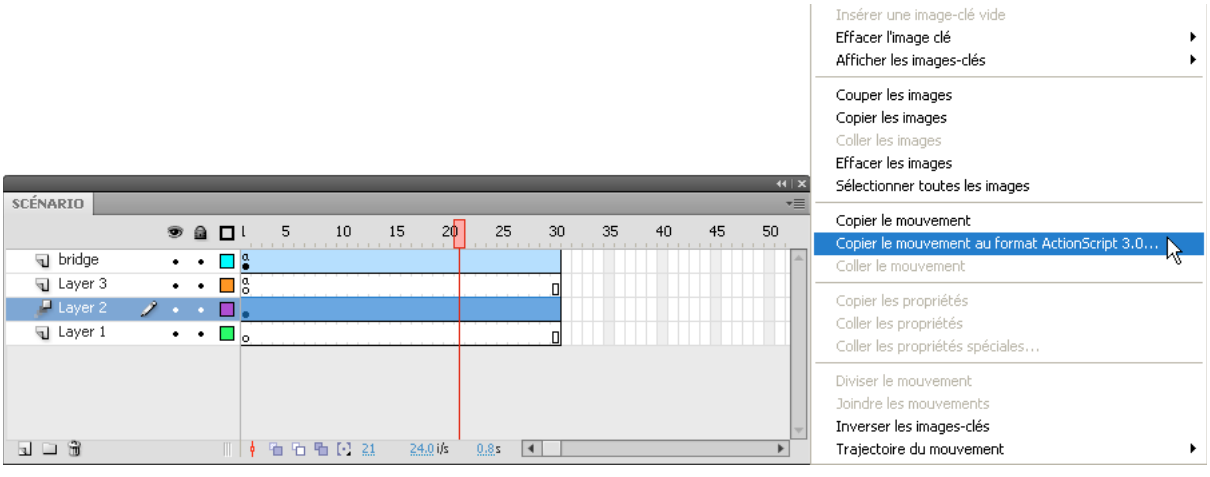
Vous créez une interpolation de mouvement dans Flash en déplaçant un objet d'affichage entre des images-clé du scénario. Flash génère automatiquement le code ActionScript qui décrit l'interpolation, que vous pouvez copier et enregistrer dans un fichier. Pour plus d'informations sur la création d'une interpolation de mouvement, consultez la section Interpolations de mouvement du manuel *Utilisation de Flash*.

Vous pouvez accéder à la copie de mouvement en tant que commande ActionScript 3.0 dans Flash, de deux façons différentes. La première consiste à utiliser un menu contextuel d'interpolation sur la scène, comme suit :

1 Sélectionnez l'interpolation de mouvement sur la scène.

2 Cliquez dessus avec le bouton droit de la souris (Windows) ou cliquez sur la touche Contrôle (Macintosh).

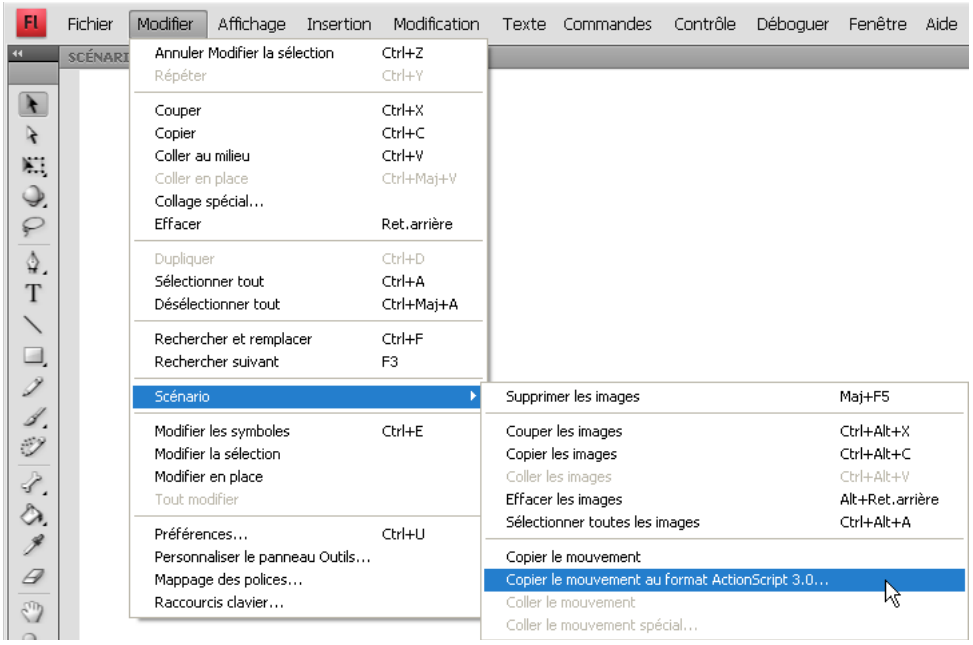
3 Choisissez Copie de mouvement en tant que ActionScript 3.0 . . .



La seconde technique consiste à choisir directement la commande dans le menu Edition de Flash, comme suit :

1 Sélectionnez l'interpolation de mouvement sur la scène.

2 Choisissez Edition > Scénario > Copier le mouvement en tant que ActionScript 3.0.



Une fois le script copié, collez-le dans un fichier et enregistrez-le.

Une fois l'interpolation de mouvement créée et après avoir copié et enregistré le script, vous pouvez la réutiliser sous sa forme actuelle ou la modifier dans votre propre animation dynamique pilotée par un script ActionScript.

Incorporation de scripts d'interpolation de mouvement

L'en-tête du code ActionScript copié à partir de Flash recense tous les modules requis pour prendre en charge l'interpolation de mouvement.

Classes d'interpolation de mouvement

Les classes principales, AnimatorFactory, MotionBase et Motion, appartiennent au package `fl.motion`. Vous pourriez avoir besoin de classes supplémentaires suivant les propriétés que l'interpolation de mouvement manipule. Par exemple, si l'interpolation de mouvement transforme ou fait pivoter l'objet d'affichage, vous devez importer les classes `flash.geom` appropriées. Si des filtres sont appliqués, importez les classes `flash.filter`. Dans ActionScript, une interpolation de mouvement est une occurrence de la classe Motion. La classe Motion stocke une séquence d'animation d'images-clés pouvant s'appliquer à un objet visuel. Les données de l'animation comprennent les éléments suivants : position, échelle, rotation, inclinaison, couleur, filtres et accélération.

Le code ActionScript suivant a été copié à partir d'une interpolation de mouvement créée en vue d'animer un objet d'affichage portant le nom d'occurrence `Symbol1_2`. Il déclare une variable associée à un objet MotionBase nommé `__motion_Symbol1_2`. La classe MotionBase est le parent de la classe Motion.

```
var __motion_Symbol1_2:MotionBase;
```

Le script crée ensuite l'objet Motion :

```
__motion_Symbol1_2 = new Motion();
```

Noms d'objet Motion

Dans le cas précédent, Flash a automatiquement généré le nom `__motion_Symbol1_2` de l'objet Motion. Il associe le préfixe `__motion_` au nom de l'objet d'affichage. Ainsi, le nom généré automatiquement est-il basé sur le nom d'occurrence de l'objet cible de l'interpolation de mouvement de l'outil de programmation Flash. La propriété `duration` de l'objet Motion indique le nombre total d'images dans l'interpolation de mouvement :

```
__motion_Symbol1_2.duration = 200;
```

Lorsque vous réutilisez ce type de code ActionScript dans votre propre animation, vous pouvez conserver le nom d'interpolation automatiquement généré par Flash, ou définir un autre nom. Par défaut, Flash affecte automatiquement un nom à l'occurrence d'objet d'affichage dont vous copiez l'interpolation de mouvement si elle n'en possède pas encore un. Si vous modifiez le nom de l'interpolation, n'oubliez pas de répercuter la modification dans l'ensemble du script. Dans Flash, vous pouvez également affecter le nom de votre choix à l'objet cible de l'interpolation de mouvement, puis créer l'interpolation de mouvement et copier le script. Quelle que soit l'approche adoptée, assurez-vous que chaque objet Motion de votre code ActionScript possède un nom unique.

Description de l'animation

La méthode `addPropertyArray()` de la classe MotionBase ajoute un tableau de valeurs pour décrire chaque propriété interpolée.

Le tableau contient potentiellement un élément par image-clé de l'interpolation de mouvement. Il arrive souvent que certains de ces tableaux contiennent moins d'éléments que le nombre total d'images-clés de l'interpolation de mouvement. Ce cas de figure se produit lorsque la dernière valeur du tableau n'est pas modifiée pour les images restantes.

Si la longueur de l'argument array est supérieure à celle de la propriété `duration` de l'objet Motion, la méthode `addPropertyArray()` ajuste en conséquence la valeur de la propriété `duration`. Elle n'ajoute pas d'image-clé pour les propriétés précédemment ajoutées. Les nouvelles images-clés subsistent pendant la durée des images supplémentaires de l'animation.

Les propriétés `x` et `y` de l'objet Motion décrivent la nouvelle position de l'objet interpolé au fur et à mesure de l'exécution de l'animation. Ces coordonnées correspondent aux valeurs les plus susceptibles de changer dans chaque image-clé si la position de l'objet d'affichage évolue. La méthode `addPropertyArray()` vous permet d'ajouter d'autres propriétés de mouvement. Par exemple, ajoutez les valeurs `scaleX` et `scaleY` si l'objet interpolé est redimensionné. Ajoutez les valeurs `skewX` et `skewY` s'il est incliné. Ajoutez la propriété `rotationConcat` s'il fait l'objet d'une rotation.

Utilisez la méthode `addPropertyArray()` pour définir les propriétés d'interpolation suivantes :

`x`	Position horizontale du point de transformation de l'objet dans l'espace de coordonnées de son objet parent
`y`	Position verticale du point de transformation de l'objet dans l'espace de coordonnées de son objet parent
`z`	Position de profondeur (axe z) du point de transformation de l'objet dans l'espace de coordonnées de son objet parent
`scaleX`	Redimensionnement horizontal, exprimé sous forme de pourcentage de l'objet tel qu'il est appliqué à partir du point de transformation
`scaleY`	Redimensionnement vertical, exprimé sous forme de pourcentage de l'objet tel qu'il est appliqué à partir du point de transformation
`skewX`	Angle d'inclinaison horizontale de l'objet, en degrés, tel qu'il est appliqué à partir du point de transformation
`skewY`	Angle d'inclinaison verticale de l'objet, en degrés, tel qu'il est appliqué à partir du point de transformation

`rotationX`	Rotation de l'objet autour de l'axe x à partir de son orientation d'origine
`rotationY`	Rotation de l'objet autour de l'axe y à partir de son orientation d'origine
`rotationConcat`	Valeurs de rotation (axe z) de l'objet dans le cadre du mouvement par rapport à l'orientation précédente appliquée à partir du point de transformation
`useRotationConcat`	Si cette propriété est définie, elle entraîne la rotation de l'objet cible lorsque la méthode `addPropertyArray()` fournit des données de mouvement.
`blendMode`	Valeur de la classe BlendMode qui définit le mélange de couleurs de l'objet sous lequel figurent les graphiques
`matrix3D`	Propriété matrix3D si elle a été définie pour l'image-clé. Réservée aux interpolations 3D. Si elle est utilisée, aucune des propriétés de transformation précédentes n'est prise en considération.
`rotationZ`	Rotation (en degrés) autour de l'axe z de l'objet, à partir de son orientation d'origine par rapport au conteneur parent 3D. Utilisé pour les interpolations 3D au lieu de rotationConcat.

Les propriétés ajoutées au script automatiquement généré varient selon les propriétés affectées à l'interpolation de mouvement dans Flash. Vous pouvez ajouter, supprimer ou modifier certaines de ces propriétés lorsque vous personnalisez votre version du script.

Le code suivant affecte des valeurs aux propriétés de l'interpolation de mouvement `__motion_Wheel`. Dans ce cas de figure, l'objet d'affichage interpolé ne change pas d'emplacement, mais pivote sur place dans les 29 images de l'interpolation de mouvement. Les diverses valeurs affectées au tableau `rotationConcat` définissent la rotation. Les autres valeurs de propriété de cette interpolation de mouvement ne sont pas modifiées.

```
__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"]);
```

Dans l'exemple suivant, l'objet d'affichage `Leaf_1` traverse la scène. Les tableaux de propriétés `x` et `y` correspondants contiennent des valeurs différentes pour chacune des 100 images de l'animation. Par ailleurs, l'objet pivote sur son axe `z` au fur et à mesure qu'il traverse la scène. Les divers éléments du tableau de la propriété `rotationZ` déterminent la rotation.

```
__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,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
    ]
);
__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"]);
```

Ajout de filtres

Si l'objet cible d'une interpolation de mouvement contient des filtres, ces derniers sont ajoutés par le biais des méthodes `initFilters()` et `addFilterPropertyArray()` de la classe Motion.

Initialisation du tableau de filtres

La méthode `initFilters()` initialise les filtres. Son premier argument est un tableau qui recense les noms de classe entièrement qualifiés de tous les filtres appliqués à l'objet d'affichage. Ce tableau de noms de filtre est généré à partir de la liste de filtres associée à l'interpolation de mouvement dans Flash. Dans votre copie du script, vous pouvez supprimer ou ajouter dans ce tableau n'importe quel filtre du package `flash.filters`. L'appel suivant initialise la liste de filtres associée à l'objet d'affichage cible. Il applique les filtres `DropShadowFilter`, `GlowFilter` et `BevelFilter` et copie la liste dans chaque image-clé de l'objet Motion.

```
__motion_Box.initFilters(["flash.filters.DropShadowFilter", "flash.filters.GlowFilter",
"flash.filters.BevelFilter"], [0, 0, 0]);
```

Ajout de filtres

La méthode `addFilterPropertyArray()` décrit les propriétés d'un filtre initialisé doté des arguments suivants :

1. Son premier argument identifie un filtre en fonction de son index. L'index se réfère à la position du nom de filtre dans le tableau des noms de classe de filtre transmis lors d'un appel précédent d'`initFilters()`.
2. Le deuxième argument est la propriété à stocker pour le filtre dans chaque image-clé.
3. Le troisième argument est la valeur de la propriété de filtre indiquée.

Etant donné l'appel précédent d'`initFilters()`, les appels suivants de `addFilterPropertyArray()` affectent la valeur 5 aux propriétés `blurX` et `blurY` de `DropShadowFilter`. `DropShadowFilter` est le premier élément (index 0)du tableau de filtres initialisés :

```
__motion_Box.addFilterPropertyArray(0, "blurX", [5]);
__motion_Box.addFilterPropertyArray(0, "blurY", [5]);
```

Les trois appels suivants affectent des valeurs aux propriétés qualité, alpha et couleur de `GlowFilter` (le deuxième élément (index 1) du tableau de filtres initialisés) :

```
__motion_Box.addFilterPropertyArray(1, "quality", [BitmapFilterQuality.LOW]);
__motion_Box.addFilterPropertyArray(1, "alpha", [1.00]);
__motion_Box.addFilterPropertyArray(1, "color", [0xff0000]);
```

Les quatre appels suivants affectent des valeurs aux propriétés `shadowAlpha`, `shadowColor`, `highlightAlpha` et `highlightColor` de `BevelFilter`, le troisième élément (index 2) du tableau de filtres initialisés :

```
__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]);
```

Réglage de la couleur à l'aide de ColorMatrixFilter

Une fois `ColorMatrixFilter` initialisé, vous pouvez définir les propriétés `AdjustColor` appropriées pour régler la luminosité, le contraste, la saturation et la teinte de l'objet d'affichage interpolé. En règle générale, vous appliquez le filtre `AdjustColor` dans Flash et vous l'ajustez dans votre copie du code ActionScript. L'exemple suivant transforme la teinte et la saturation de l'objet d'affichage au fur et à mesure qu'il se déplace.

```
__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);
```

Association d'une interpolation de mouvement à ses objets d'affichage

La dernière tâche consiste à associer l'interpolation de mouvement aux objets d'affichage qu'elle manipule.

La classe AnimatorFactory gère l'association entre une interpolation de mouvement et ses objets d'affichage cible. L'argument du constructeur d'AnimatorFactory correspond à l'objet Motion :

```
var __animFactory_Wheel:AnimatorFactory = new AnimatorFactory(__motion_Wheel);
```

La méthode `addTarget()` de la classe AnimatorFactory permet d'associer l'objet d'affichage cible à l'interpolation de mouvement correspondante. Le code ActionScript copié à partir de Flash met la ligne `addTarget()` en commentaire et n'indique pas de nom d'occurrence :

```
// __animFactory_Wheel.addTarget(<instance name goes here>, 0);
```

Dans votre copie, stipulez l'objet d'affichage à associer à l'interpolation de mouvement. Dans l'exemple suivant, les cibles spécifiées sont `greenWheel` et `redWheel` :

```
__animFactory_Wheel.AnimatorFactory.addTarget(greenWheel, 0);
__animFactory_Wheel.AnimationFactory.addTarget(redWheel, 0);
```

Vous pouvez associer plusieurs objets d'affichage à la même interpolation de mouvement en utilisant plusieurs appels de `addTarget()`.

Chapitre 20 : Utilisation de la cinématique inverse

La cinématique inverse (IK, Inverse kinematics) est une technique de création fantastique de mouvement réaliste.

IK vous permet de créer des mouvements coordonnés au sein d'une chaîne de sections connectées appelée squelette IK, de sorte que les sections se déplacent avec réalisme. Les sections du squelette représentent ses os et articulations. A partir de l'extrémité finale du squelette, IK calcule les angles des articulations requises pour atteindre cette dernière.

Calculer manuellement ces angles s'avérerait particulièrement complexe. Cette fonctionnalité présente l'avantage de permettre de créer des squelettes en mode interactif dans Adobe® Flash® CS4 Professional. Il vous suffit ensuite de les animer par le biais d'ActionScript. Le moteur IK contenu dans l'outil de programmation Flash exécute les calculs pour décrire le mouvement du squelette. Vous pouvez restreindre le mouvement à certains paramètres dans votre code ActionScript.

Pour créer des squelettes de cinématique inverse, vous devez disposer d'une licence pour Adobe Flash CS4 Professional.

Principes de base de la cinématique inverse

Introduction à IK

La cinématique inverse (IK) vous permet de créer une animation réaliste en liant des sections de sorte qu'elles se déplacent les unes par rapport aux autres avec naturel.

L'utilisation d'IK vous permet par exemple de déplacer une jambe pour qu'elle occupe une position déterminée en articulant les mouvements des articulations de la jambe nécessaires à l'obtention de la pose appropriée. IK utilise une structure osseuse chaînée portant le nom de squelette IK. Le package `fl.ik` vous aide à créer des animations qui ressemblent à un mouvement naturel. Il vous permet d'animer plusieurs squelettes IK en toute transparence sans avoir à maîtriser les concepts physiques sur lesquels s'appuient les algorithmes IK.

Vous créez le squelette IK et les os et articulations qui le composent dans Flash. Vous pouvez ensuite utiliser les classes IK pour les animer lors de l'exécution.

Pour obtenir des instructions détaillées sur la création d'un squelette IK, consultez la section Utilisation de la cinématique inverse dans le manuel *Utilisation de Flash CS4 Professional*.

Tâches IK courantes

Votre code ActionScript de lancement et de contrôle du mouvement d'un squelette IK lors de l'exécution effectue généralement les tâches suivantes :

- Il déclare les variables relatives aux squelettes, aux os et aux articulations impliqués dans le mouvement.
- Il extrait les occurrences de squelette, d'os et d'articulation.
- Il instancie l'objet IKMover.
- Il définit les restrictions du mouvement.
- Il déplace le squelette de sorte qu'il occupe un point cible.

Terminologie et concepts importants

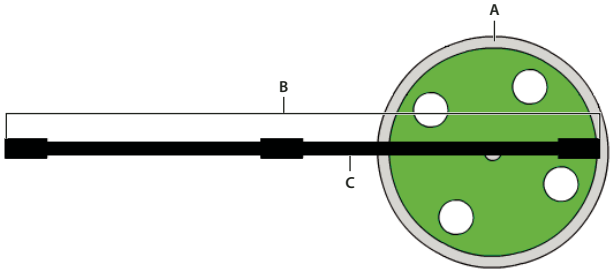
La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Squelette chaîne cinématique composée d'os et d'articulations, utilisée en animation informatique pour simuler un mouvement réaliste.
- Os : segment rigide d'un squelette, équivalent à un os chez un animal.
- Cinématique inverse (IK) : processus d'identification des paramètres d'un objet souple articulé appelé squelette ou chaîne cinématique.
- Articulation : emplacement où deux os s'unissent, conçu pour permettre le mouvement des os ; analogue à une articulation chez un animal.

Aperçu de l'animation de squelettes IK

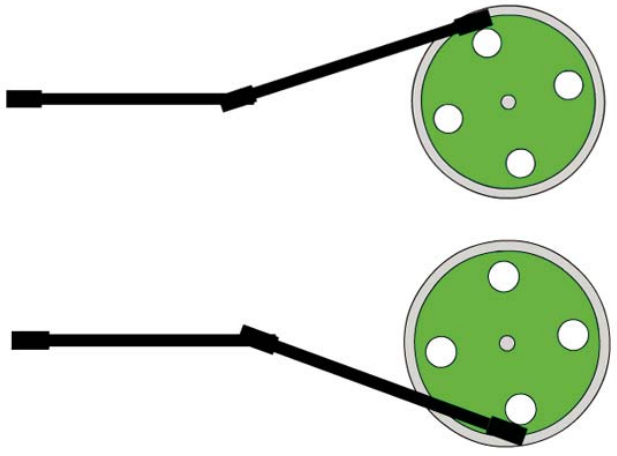
Une fois le squelette IK créé, utilisez les classes `fl.ik` pour restreindre ses mouvements, suivre les événements correspondants et l'animer lors de l'exécution.

La figure ci-dessous illustre le clip `Wheel`. L'essieu est une occurrence d'un squelette IK appelée `Axle`. La classe IKMover déplace le squelette en le synchronisant avec la rotation d'une roue. Dans le squelette, IKBone, nommé `ikBone2`, est rattaché à la roue au niveau de l'articulation arrière.



A. *Roue* ***B.*** *Essieu* ***C.*** *ikBone2*

Lors de l'exécution, la roue tourne en association avec l'interpolation de mouvement `__motion_Wheel` décrite à la section « Description de l'animation » à la page 432 du chapitre Utilisation des interpolations de mouvement. Un objet IKMover lance et contrôle le mouvement de l'essieu. La figure suivante propose deux instantanés du squelette de l'essieu connecté à la roue qui tourne sur différentes images de la rotation.



Lors de l'exécution, le code ActionScript suivant :

- Extrait des informations relatives au squelette et à ses composants.
- Instancie un objet IKMover.
- Déplace l'essieu en conjonction avec la rotation de la roue.

```
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);
    }
}
```

Les classes IK utilisées pour déplacer l'essieu sont les suivantes :

- IKArmature : décrit le squelette (structure arborescente composée d'os et d'articulations). A créer dans Flash
- IKManager : classe qui contient tous les squelettes IK du document, à créer dans Flash
- IKBone : segment d'un squelette IK.
- IKJoint : connexion entre deux os IK.
- IKMover : lance et contrôle le mouvement IK des squelettes.

Pour obtenir une description détaillée de ces classes, consultez le package ik.

Obtention d'informations sur un squelette IK

Commencez par déclarer les variables associées au squelette, à l'os et à l'articulation qui composent les sections à déplacer.

Le code suivant utilise la méthode `getsqueletteByName()` de la classe IKManager pour affecter la valeur du squelette Axle à la variable IKArmature `tree`. Le squelette Axle a été précédemment généré dans Flash.

```
var tree:IKArmature = IKManager.getArmatureByName("Axle");
```

De même, le code suivant utilise la méthode `getBoneByName()` de la classe IKArmature pour affecter à la variable IKBone la valeur de l'os `ikBone2`.

```
var bone:IKBone = tree.getBoneByName("ikBone2");
```

L'articulation arrière de l'os `ikBone2` correspond à la section du squelette connectée à la roue qui tourne.

La ligne suivante déclare la variable `endEffector` et l'affecte à la propriété `tailjoint` de l'os `ikBone2` :

```
var endEffector:IKJoint = home.tailjoint;
```

La variable `pos` est un point qui stocke la position actuelle de l'articulation `endEffector`.

```
var pos:Point = endEffector.position;
```

Dans cet exemple, `pos` correspond à la position de l'articulation raccordée à la roue à l'extrémité de l'essieu. La valeur d'origine de cette variable est extraite de la propriété `position` de IKJoint.

Instanciation de l'objet IKMover et restriction du mouvement

Une occurrence de la classe IKMover déplace l'essieu.

La ligne suivante instancie l'objet IKMover `ik` et transmet à son constructeur l'élément à déplacer et le point de départ du mouvement :

```
var ik:IKMover = new IKMover(endEffector, pos);
```

Les propriétés de la classe IKMover vous permettent de restreindre le mouvement d'un squelette. Vous pouvez restreindre le mouvement en fonction de la distance, des itérations et de la durée du mouvement.

Les paires de propriétés suivantes imposent ces restrictions. Les paires se composent d'une valeur `booléenne` qui indique si le mouvement est restreint et d'un nombre stipulant la restriction :

`limitByDistance:Boolean`	`distanceLimit:int`	Définit la distance maximale en pixels parcourue par le moteur IK par itération.
`limitByIteration:Boolean`	`iterationLimit:int`	Définit le nombre maximal d'itérations effectuées par le moteur IK par mouvement.
`limitByTime:Boolean`	`timeLimit:int`	Définit la durée maximale, exprimée en millisecondes, affectée au moteur IK pour effectuer le mouvement.

Définissez la propriété `Boolean` appropriée sur `true` pour imposer la restriction. A moins que vous ne les définissiez explicitement sur « true », toutes les propriétés `Boolean` sont définies par défaut sur `false` pour ne pas restreindre le mouvement. Si vous définissez la restriction sur une valeur sans définir la propriété `Boolean` correspondante, elle n'est pas prise en considération. Dans ce cas, le moteur IK continue à déplacer l'objet jusqu'à ce qu'une autre restriction ou la position cible de l'objet IKMover soit atteinte.

Dans l'exemple suivant, la distance maximale parcourue par le mouvement du squelette est définie sur 0,1 pixel par itération. Le nombre maximal d'itérations par mouvement est défini sur dix.

```
ik.limitByDistance = true;
ik.distanceLimit = 0.1;
ik.limitByIteration = true;
ik.iterationLimit = 10;
```

Mouvement d'un squelette IK

L'objet IKMover déplace l'essieu au sein de l'écouteur d'événement associé à la roue. A chaque événement enterFrame de la roue, une nouvelle position cible du squelette est calculée. Par le biais de sa méthode `moveTo()` , l'objet IKMover place l'articulation arrière sur sa position cible ou parcourt une distance aussi longue que possible sans enfreindre les contraintes définies par ses propriétés `limitByDistance`, `limitByIteration` et `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);
    }
}
```

Utilisation d'événements IK

La classe IKEvent vous permet de créer un objet événement qui contient des informations sur les événements IK. Une information IKEvent décrit le mouvement qui s'est arrêté car la durée, la distance ou le nombre maximal d'itérations stipulés ont été dépassés.

Le code suivant indique un écouteur et un gestionnaire d'événement destinés au suivi des événements de limite de temps. Ce gestionnaire d'événement signale les propriétés de durée, distance, nombre d'itérations et articulations d'un événement déclenché lorsque la durée maximale de l'objet IKMover est dépassée.

```
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);
}
```

Chapitre 21 : Utilisation de texte

Pour afficher du texte à l'écran dans Adobe® Flash® Player ou Adobe® AIR™, utilisez une occurrence de la classe TextField ou les classes Text Engine de Flash. Ces classes vous permettent de créer, d'afficher et de mettre en forme du texte.

Vous pouvez établir le contenu spécifique de champs de texte ou désigner la source du texte, puis en définir l'aspect. Vous pouvez également répondre aux événements utilisateur (saisie de texte ou clic sur un lien hypertexte).

Principes de base de l'utilisation du texte

Introduction à l'utilisation du texte

La classe TextField et les classes Flash Text Engine vous permettent d'afficher et de gérer le texte dans Flash Player et AIR.

Vous pouvez utiliser la classe TextField pour créer des objets texte à des fins d'affichage et de saisie. Cette classe est le fondement d'autres composants à base de texte, tels que TextArea et TextInput, que proposent Flash et Adobe Flex. La classe TextFormat permet de définir la mise en forme de caractère et paragraphe des objets TextField et vous pouvez appliquer des feuilles de style en cascade (CSS) à l'aide de la propriété Textfield.styleSheet et de la classe StyleSheet. Vous pouvez affecter directement à un champ de texte un texte au format HTML, qui peut contenir des médias intégrés (clips, fichiers SWF, fichiers GIF, fichiers PNG et fichiers JPEG).

Flash Text Engine (FTE), disponible à partir de Flash Player 10 et Adobe AIR 1.5, propose une prise en charge de bas niveau pour un contrôle sophistiqué des mesures de texte, de la mise en forme et du texte bidirectionnel. Il se caractérise également par un flux de texte optimisé et une prise en charge des langues enrichie. Vous pouvez utiliser Flash Text Engine pour créer et gérer des éléments de texte, mais c'est surtout l'outil de base pour créer des composants de gestion de texte et, à ce titre, il exige des compétences de programmation avancées.

Tâches courantes d'utilisation du texte

Les tâches courantes suivantes relatives à la classe TextField sont traitées :

- Modification du contenu d'un champ de texte
- Utilisation de HTML dans des champs de texte
- Utilisation d'images dans des champs de texte
- Sélection de texte et actions sur le texte sélectionné par l'utilisateur
- Capture du texte saisi par l'utilisateur
- Restriction de la saisie de texte
- Application d'une mise en forme et de styles CSS à du texte
- Contrôle de la netteté, de l'épaisseur et du lissage
- Manipulation de champs de texte statique en ActionScript

Les tâches courantes suivantes relatives aux classes Flash Text Engine sont traitées :

- Création et affichage de texte

- Gestion des événements
- Mise en forme du texte
- Utilisation des polices
- Contrôle du texte

Concepts importants et terminologie

La liste de référence suivante énumère les termes importants que vous rencontrerez :

- Feuilles de style en cascade : syntaxe standardisée permettant de définir des styles et une mise en forme pour du texte structuré en XML et en HTML.
- Police de périphérique : police installée sur l'ordinateur de l'utilisateur.
- Champ de texte dynamique : champ de texte dont le contenu peut être modifié en ActionScript, mais pas par l'utilisateur.
- Police intégrée : police de caractères dont les données, sous forme vectorielle, sont enregistrées dans le fichier SWF de l'application.
- Texte HTML : texte inséré dans un champ de texte en ActionScript et comportant des balises HTML de mise en forme, outre le contenu lui-même.
- Champ de saisie de texte : champ de texte dont le contenu peut être modifié soit en ActionScript, soit par l'utilisateur.
- Crénage : réglage de l'espace entre les paires de caractères de sorte à uniformiser l'espacement des mots et à améliorer la lisibilité du texte.
- Champ de texte statique : champ de texte créé dans l'environnement de programmation, et dont le contenu ne peut pas être modifié pendant l'exécution du fichier SWF.
- Métrique des lignes de texte : mesure de la taille des diverses parties du texte contenu dans un champ de texte: ligne de base du texte, hauteur du sommet des caractères, taille des jambages (la partie de certaines minuscules qui s'étend sous la ligne de base), etc.
- Interlettrage : réglage de l'espacement entre des groupes de lettres ou des blocs de texte en vue d'augmenter ou de réduire la densité pour améliorer la lisibilité du texte.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester des exemples de code. Pour tester un exemple de code, il est en général nécessaire de manipuler un objet texte (créé et placé sur la scène dans l'outil de programmation Flash, ou créé à l'aide d'ActionScript). Pour tester un exemple, il est nécessaire d'afficher le résultat dans Flash Player ou AIR afin de voir l'effet du code sur le champ de texte ou l'objet TextField.

Les exemples de cette rubrique peuvent être classés en deux groupes : Dans le premier type d'exemples, l'objet TextField est manipulé sans être explicitement créé. Pour tester ces exemples de code, procédez comme suit :

1. Créez un document Flash vide.
2. Sélectionnez une image-clé dans le scénario.
3. Ouvrez le panneau Actions et copiez le code dans le panneau Script.
4. Créez un champ de texte dynamique sur la Scène à l'aide de l'outil Texte.

5 Sélectionnez le champ de texte, puis donnez-lui un nom d'occurrence dans l'Inspecteur des propriétés. Le nom doit correspondre au nom utilisé pour le champ de texte dans l'exemple de code (par exemple, si le code manipule un champ de texte `myTextField`, vous devez appeler votre champ de texte `myTextField` également).

6 Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

Les résultats du code manipulant le champ de texte s'affichent à l'écran, comme indiqué dans le code.

L'autre type d'exemples de code consiste en une définition de classe censée être utilisée comme classe de document pour le fichier SWF. Dans ces exemples, l'exemple de code crée les objets texte ; il n'est donc pas nécessaire de les créer séparément. Pour tester ce type d'exemples de code :

1 Créez un document Flash vide et enregistrez-le sur votre ordinateur.

2 Créez un nouveau fichier ActionScript et enregistrez-le dans le même répertoire que le document Flash. Le nom du fichier doit correspondre au nom de la classe du code. Par exemple, si le code définit une classe TextFieldTest, enregistrez le fichier ActionScript sous le nom TextFieldTest.as.

3 Copiez le code dans le fichier ActionScript et enregistrez le fichier.

4 Dans le document Flash, sélectionnez Fenêtre > Propriétés pour activer l'Inspecteur des propriétés du document.

5 Dans l'Inspecteur des propriétés, dans le champ Classe, saisissez le nom de la classe ActionScript que vous avez copiée du texte.

6 Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

Les résultats de l'exemple s'affichent à l'écran.

D'autres techniques de test d'exemples de code sont expliquées plus en détail à la section « Test des exemples de code contenus dans un chapitre » à la page 36.

Utilisation de la classe TextField

La classe TextField est la base des autres composants dévolus au texte (par exemple les composants TextArea ou TextInput) dans la structure Adobe Flex et l'environnement de programmation Flash. Pour plus d'informations sur l'utilisation des composants de texte dans l'environnement de programmation Flash, consultez la section « A propos des commandes de texte » du guide *Utilisation de Flash*.

Le contenu des champs de texte peut être spécifié à l'avance dans le fichier SWF, chargé à partir d'un fichier texte ou d'une base de données, ou saisi par l'utilisateur dans votre application. Au sein du champ lui-même, le texte peut être du contenu HTML, avec des images incorporées. Après avoir créé une occurrence de champ de texte,vous pouvez utiliser les classes flash.text, telles que TextFormat et StyleSheet, pour contrôler l'aspect du texte. Le package flash.text contient la plupart des classes relatives à la création, la gestion et la mise en forme de texte dans ActionScript.

Pour mettre en forme du texte, il est nécessaire de créer un objet TextFormat et de l'affecter au champ de texte. Si le champ de texte contient du texte en HTML, vous pouvez lui appliquer un objet StyleSheet pour affecter des styles à des éléments spécifiques du texte. L'objet TextFormat ou StyleSheet contient des propriétés qui définissent l'aspect du texte, par exemple sa couleur, sa taille et sa graisse. L'objet TextFormat attribue des propriétés à l'ensemble du contenu d'un champ de texte, ou à une partie du texte seulement. Par exemple, au sein du même champ de texte, une phrase peut être en gras et en rouge, puis la suivante en italique et en bleu.

Pour plus d'informations sur les formats de texte, consultez la section « Attribution de formats texte » à la page 452.

Pour plus d'informations sur le texte HTML dans les champs de texte, consultez la section « Affichage du texte HTML » à la page 447.

Pour plus d'informations sur les feuilles de style, consultez la section « Application de feuilles de style en cascade » à la page 453.

Outre les classes du package flash.text, la classe flash.events.TextEvent permet de répondre aux actions de l'utilisateur liées au texte.

Affichage du texte

Bien que les outils de programmation tels qu'Adobe Flex Builder et Flash offrent plusieurs options d'affichage du texte (composants liés au texte ou outils texte) la principale méthode d'affichage de texte par programmation passe par un champ de texte.

Types de texte

Le type de texte d'un champ de texte est caractérisé par sa source :

- Texte dynamique

  Le texte dynamique correspond au contenu chargé à partir d'une source externe, telles qu'un fichier texte ou xml, ou un service Web.

- Saisie de texte

  Le texte saisi est le texte entré par l'utilisateur ou du texte dynamique que l'utilisateur peut modifier. Vous pouvez définir une feuille de style pour formater le texte saisi, ou utiliser la classe flash.text.TextFormat pour attribuer au champ de texte des propriétés destinées au texte saisi. Pour plus d'informations, consultez la section « Capture du texte saisi par l'utilisateur » à la page 450.

- Texte statique

  Le texte statique peut uniquement être créé par le biais de l'outil de programmation Vous ne pouvez pas créer une occurrence de texte à l'aide d'ActionScript 3.0. Vous pouvez néanmoins utiliser les classes ActionScript telles que StaticText et TextSnapshot pour manipuler une occurrence de texte statique existante. Pour plus d'informations, consultez la section « Utilisation du texte statique » à la page 457.

Modification du contenu d'un champ de texte

Vous pouvez définir du texte dynamique en affectant une chaîne à la propriété `flash.text.TextField.text`. La chaîne est directement affectée à la propriété, comme suit :

```
myTextField.text = "Hello World";
```

Vous pouvez également affecter à la propriété `text` une valeur issue d'une variable définie dans votre code, comme dans l'exemple suivant :

```
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;
        }
    }
}
```

Vous pouvez également attribuer à la propriété `text` une valeur issue d'une variable distante. Le chargement de valeurs textuelles à partir de sources distantes peut se faire de trois manières :

- Les classes flash.net.URLLoader et flash.net.URLRequest chargent des variables à partir d'emplacements locaux ou distants.
- L'attribut `FlashVars` est incorporé dans la page HTML qui héberge le fichier SWF et peut contenir des valeurs destinées aux variables de texte.
- La classe flash.net.SharedObject gère le stockage persistant des valeurs. Pour plus d'informations, consultez la section « Stockage des données locales » à la page 637.

Affichage du texte HTML

La propriété `htmlText` de la classe flash.text.TextField permet d'indiquer que la chaîne de texte contient des balises HTML de formatage du contenu. Comme le montre l'exemple suivant, vous devez affecter votre chaîne à la propriété `htmlText` (et non à la propriété `text`) pour que Flash Player ou AIR puisse afficher le texte sous forme HTML :

```
var myText:String = "<p>This is <b>some</b> content to <i>render</i> as <u>HTML</u> text.</p>";
myTextBox.htmlText = myText;
```

Pour la propriété `htmlText`, Flash Player et AIR prennent en charge un sous-ensemble de balises et d'entités HTML. La description de propriété `flash.text.TextField.htmlText` dans le Guide de référence du langage et des composants ActionScript 3.0 fournit des informations détaillées sur les balises et entités HTML prises en charge.

Une fois que vous avez spécifié votre contenu à l'aide de la propriété `htmlText`, vous pouvez utiliser des feuilles de style ou la balise `textformat` pour gérer le formatage. Pour plus d'informations, consultez la section « Mise en forme du texte » à la page 452.

Utilisation d'images dans des champs de texte

L'affichage du contenu sous forme de texte HTML présente un autre avantage : vous pouvez inclure des images dans le champ de texte. Il est possible de référencer une image, locale ou distante, grâce à la balise `img` et de la faire apparaître dans le champ de texte associé.

L'exemple suivant crée un champ de texte appelé `myTextBox` et incorpore au texte une image JPG représentant un œil, image stockée dans le même répertoire que le fichier SWF :

```
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;
        }
    }
}
```

La balise `img` prend en charge les fichiers JPEG, GIF, PNG et SWF.

Défilement du texte dans un champ de texte

Dans bien des cas, votre texte peut s'avérer plus long que le champ de texte qui le contient. Il se peut également qu'un champ de saisie permette à l'utilisateur de saisir plus de caractères qu'il ne peut en afficher en une seule fois. Les propriétés de défilement de la classe flash.text.TextField permettent de gérer du contenu long, que ce soit verticalement ou horizontalement.

Ces propriétés sont les suivantes : `TextField.scrollV`, `TextField.scrollH`, `maxScrollV` et `maxScrollH`. Utilisez-les pour répondre à des événements tels qu'un clic de souris ou une pression sur une touche.

L'exemple ci-après crée un champ de texte de taille fixe et contenant plus de texte que le champ ne peut afficher en une seule fois. Lorsque l'utilisateur clique sur le champ de texte, le texte défile verticalement.

```
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++;
        }
    }
}
```

Sélection et manipulation de texte

Vous pouvez sélectionner du texte, qu'il soit dynamique ou saisi. Les propriétés et méthodes de sélection de texte de la classe TextField utilisent des positions d'index pour déterminer l'étendue du texte à manipuler. Vous pouvez donc programmer la sélection du texte saisi ou dynamique, même si vous n'en connaissez pas le contenu.

Remarque : *dans l'outil de programmation Flash, si vous choisissez l'option sélectionnable pour un champ de texte statique, le champ de texte exporté et placé dans la liste d'affichage est un champ de texte dynamique normal.*

Sélection du texte

La propriété `flash.text.TextField.selectable` a la valeur `true` par défaut. Vous pouvez en outre sélectionner du texte par code à l'aide de la méthode `setSelection()`.

Par exemple, pour sélectionner un texte spécifique dans un champ de texte lorsque l'utilisateur clique dans ce dernier :

```
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);
}
```

De même, pour que le texte d'un champ de texte soit sélectionné dès son affichage initial, créez une fonction de gestion d'événement qui sera appelée lorsque le champ de texte sera ajouté à la liste d'affichage.

Capture du texte sélectionné par l'utilisateur

Les propriétés `selectionBeginIndex` et `selectionEndIndex` de TextField, qui sont en lecture seule (et ne peuvent donc pas être utilisées à l'aide de code pour sélectionner du texte), permettent également de capturer la sélection actuelle effectuée par l'utilisateur. Par ailleurs, les champs de texte saisis peuvent utiliser la propriété `caretIndex`.

Par exemple, ce code renvoie les valeurs d'index du texte sélectionné par l'utilisateur :

```
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);
}
```

Vous pouvez également appliquer un ensemble de propriétés de l'objet TextFormat à la sélection pour modifier l'aspect du texte. Pour plus d'informations sur l'application d'un ensemble de propriétés TextFormat au texte sélectionné, consultez la section « Formatage de plages de texte au sein d'un champ de texte » à la page 455.

Capture du texte saisi par l'utilisateur

Par défaut, la propriété `type` d'un champ de texte est définie sur `dynamic`. Si vous attribuez à cette propriété `type` la valeur `input` à l'aide de la classe TextFieldType, vous pouvez recueillir la saisie de l'utilisateur et enregistrer cette valeur pour l'utiliser dans d'autres zones de l'application. Les champs de texte saisi sont utiles dans les formulaires et toute autre application qui attend que l'utilisateur définisse une valeur de texte à utiliser ailleurs dans le programme.

Par exemple, le code suivant crée un champ de texte de saisie appelé `myTextBox`. Lorsque l'utilisateur saisit du texte dans le champ, l'événement `textInput` est déclenché. Un gestionnaire d'événement appelé `textInputCapture` capture la chaîne de texte saisie et l'attribue à une variable. Flash Player ou AIR affiche le nouveau texte dans un autre champ de texte appelé `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;
        }

    }
}
```

Restriction de la saisie de texte

Les champs de texte de saisie sont souvent utilisés dans les formulaires et les boîtes de dialogue des applications. Il peut donc être judicieux de limiter le type de caractères que l'utilisateur peut saisir, ou même de masquer la saisie (pour un mot de passe par exemple). La classe flash.text.TextField possède une propriété `displayAsPassword` et une propriété `restrict` qui permettent de contrôler la saisie par l'utilisateur.

La propriété `displayAsPassword` masque simplement le texte (en l'affichant sous forme d'astérisques) à mesure que l'utilisateur le saisit. Lorsque `displayAsPassword` a la valeur `true`, les commandes Couper et Copier, ainsi que les raccourcis clavier correspondants ne fonctionnent pas. Comme le montre l'exemple suivant, vous pouvez attribuer la propriété `displayAsPassword`, comme vous le feriez pour des propriétés d'arrière-plan et de couleur :

```
myTextBox.type = TextFieldType.INPUT;
myTextBox.background = true;
myTextBox.displayAsPassword = true;
addChild(myTextBox);
```

La propriété `restrict` est légèrement plus compliquée, puisque vous devez spécifier les caractères que l'utilisateur peut saisir dans le champ de texte. Il est possible d'autoriser la saisie de lettres spécifiques et de nombres, mais aussi de plages de lettres, de nombres et de caractères. Le code ci-après permet à l'utilisateur de saisir uniquement des lettres majuscules (pas de nombres, ni de caractères spéciaux) dans le champ de texte :

```
myTextBox.restrict = "A-Z";
```

ActionScript 3.0 utilise le tiret pour définir les séries et le caractère circonflexe pour exclure des caractères. Pour plus d'informations sur la définition de restrictions pour un champ de texte de saisie, reportez-vous à l'entrée `flash.text.TextField.restrict` dans le Guide de référence du langage et des composants ActionScript 3.0.

Mise en forme du texte

Plusieurs options permettent de programmer la mise en forme du texte à afficher. Vous pouvez définir ses propriétés directement dans l'occurrence de TextField, par exemple `TextFIeld.thickness`, `TextField.textColor` et `TextField.textHeight`.Vous pouvez aussi désigner le contenu du champ de texte à l'aide de la propriété `htmlText` et utiliser des balises HTML prises en charge, telles que `b`, `i` et `u`. Vous pouvez enfin appliquer des objets TextFormat aux champs de texte contenant du texte brut, ou des objets StyleSheet aux champs contenant la propriété `htmlText`. Les objets TextFormat et StyleSheet offrent un meilleur contrôle et davantage de cohérence sur l'aspect du texte pour l'ensemble de l'application. Il est possible de définir un objet TextFormat ou StyleSheet et de l'appliquer à une partie ou à l'ensemble des champs de texte de l'application.

Attribution de formats texte

La classe TextFormat permet de définir différentes propriétés d'affichage du texte et de les appliquer à tout le contenu d'un objet TextField, ou à une plage de texte.

L'exemple suivant applique un objet TextFormat à un objet TextField complet, puis un second objet TextFormat à une plage de texte de cet objet TextField :

```
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);
```

La méthode `TextField.setTextFormat()` n'affecte que le texte qui est déjà affiché dans le champ de texte. Si le contenu de l'objet TextField change, il peut être nécessaire d'appeler à nouveau la méthode `TextField.setTextFormat()` pour ré-appliquer la mise en forme. Vous pouvez également utiliser la propriété `defaultTextFormat` de TextField pour spécifier le format à utiliser pour le texte saisi par l'utilisateur.

Application de feuilles de style en cascade

Les champs de texte peuvent contenir du texte brut ou du texte au format HTML. Le texte brut est stocké dans la propriété `text` de l'occurrence, et le texte HTML dans la propriété `htmlText`.

Vous pouvez utiliser des déclarations de styles CSS pour définir des styles de texte à appliquer ensuite à différents champs de texte. Une déclaration de style CSS peut être créée par code ou chargée lors de l'exécution à partir d'un fichier CSS externe.

C'est la classe flash.text.StyleSheet qui gère les styles CSS. La classe StyleSheet ne reconnaît qu'un nombre limité de propriétés CSS. La liste des propriétés de style prises en charge par la classe StyleSheet figure dans l'entrée flash.textStylesheet du Guide de référence du langage et des composants ActionScript 3.0.

Comme le montre l'exemple suivant, vous pouvez créer des feuilles de style CSS et les appliquer à du texte HTML au moyen de l'objet StyleSheet :

```
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);
```

Après la création de l'objet StyleSheet, le code crée un objet simple pour contenir un jeu de propriétés de déclaration de style. Il appelle ensuite la méthode `StyleSheet.setStyle()`, qui ajoute le nouveau style à la feuille de style sous le nom « .darkred ». Puis il applique les formats de la feuille de styles en affectant l'objet StyleSheet à la propriété `styleSheet` de TextField.

Pour que les styles CSS puissent prendre effet, il est nécessaire d'appliquer la feuille de style à l'objet TextField avant de définir la propriété `htmlText`.

Par essence, un champ de texte doté d'une feuille de style n'est pas modifiable. Si vous attribuez une feuille de style à un champ de texte de saisie, le champ de texte affiche les propriétés de la feuille de style, mais le champ de texte ne permet pas à l'utilisateur de saisir du texte. En outre, vous ne pouvez pas utiliser les méthodes ActionScript suivantes sur un champ de texte doté d'une feuille de style :

- La méthode `TextField.replaceText()`
- La méthode `TextField.replaceSelectedText()`
- La propriété `TextField.defaultTextFormat`
- La méthode `TextField.setTextFormat()`

Si un champ de texte est doté d'une feuille de style mais que par la suite la propriété `TextField.styleSheet` reçoit la valeur `null`, `TextField.text` et `TextField.htmlText` ajoutent des balises et des attributs à leurs contenus afin d'incorporer le formatage de la feuille de style précédemment attribuée. Pour préserver la propriété `htmlText` d'origine, enregistrez-la dans une variable avant d'attribuer la valeur `null` à la feuille de style.

Chargement de fichiers CSS externes

L'utilisation de feuilles de style CSS pour la mise en forme offre plus de possibilités s'il est possible de charger les informations de CSS à partir d'un fichier externe lors de l'exécution. Si les données CSS sont externes à l'application, il est possible de changer le style visuel du texte sans devoir modifier le code source ActionScript 3.0. En effet, après le déploiement de l'application, vous pouvez encore modifier le fichier CSS externe pour obtenir un nouvel aspect, sans devoir redéployer le fichier SWF de l'application.

La méthode `StyleSheet.parseCSS()` convertit une chaîne contenant des données CSS en déclarations de style dans l'objet StyleSheet. L'exemple suivant montre comment lire un fichier CSS externe et appliquer ses déclarations de style à un objet TextField.

Voici le contenu du fichier CSS à charger. Il est appelé « 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;
}
```

Voici maintenant le code ActionScript d'une classe qui charge le fichier example.css et en applique les styles au contenu de l'objet TextField :

```
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;
        }
    }
}
```

Lorsque les données de CSS sont chargées, la méthode `onCSSFileLoaded()` s'exécute et appelle la méthode `StyleSheet.parseCSS()` pour transférer les déclarations de style à l'objet StyleSheet.

Formatage de plages de texte au sein d'un champ de texte

La classe flash.text.TextField contient une méthode particulièrement utile, `setTextFormat()`. La méthode `setTextFormat()` permet d'affecter des propriétés spécifiques à une partie du contenu d'un champ de texte en réponse à une action de l'utilisateur, par exemple pour rappeler à l'utilisateur que certains champs d'un formulaire doivent être renseignés, ou encore pour changer l'aspect d'un passage de texte si l'utilisateur sélectionne une partie de ce texte.

L'exemple suivant utilise la méthode `TextField.setTextFormat()` sur une plage de caractères pour modifier l'aspect d'une partie du contenu de `myTextField` lorsque l'utilisateur clique dans ce champ de texte :

```
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);
}
```

Fonctions avancées d'affichage de texte

Le package flash.text d'ActionScript 3.0 offre plusieurs classes qui permettent de contrôler les propriétés du texte affiché, notamment les polices intégrées, les paramètres de lissage, le canal alpha et autres paramètres spécifiques. Le Guide de référence du langage et des composants ActionScript 3.0 fournit des descriptions détaillées de ces classes et de leurs propriétés, notamment des classes CSMSettings, Font et TextRenderer.

Utilisation de polices incorporées

Si vous spécifiez une police précise pour un objet TextField de votre application, Flash Player ou AIR recherche une police résidente du même nom sur l'ordinateur de l'utilisateur. Si cette police n'est pas chargée sur cet ordinateur, ou s'il existe une police de ce nom mais dans une version légèrement différente, le texte peut apparaître très différent de ce que vous aviez prévu.

Pour que l'utilisateur voie exactement la police voulue, vous pouvez incorporer cette police dans le fichier SWF de votre application. Les polices intégrées présentent de nombreux avantages :

- Les caractères des polices incorporées sont lissés, ce qui les rend plus agréables à lire, en particulier pour les grandes tailles de texte.
- Il est possible de faire pivoter les polices incorporées.
- Il est possible de rendre transparent ou semi-transparent le texte des polices incorporées.
- Il est possible d'utiliser le style CSS `kerning` (crénage) avec les polices incorporées.

Le principal inconvénient des polices incorporées est l'augmentation de la taille du fichier de l'application.

La méthode exacte à utiliser pour intégrer un fichier de police dans le fichier SWF de l'application varie selon l'environnement de développement.

Une fois la police intégrée, il est possible de faire en sorte que l'objet TextField utilise la police correcte :

- Mettez la propriété `embedFonts` de l'objet TextField sur `true`.
- Créez un objet TextFormat, donnez à sa propriété `fontFamily` le nom de la police incorporée, et appliquez l'objet TextFormat au TextField. Dans le cas d'une police incorporée, la propriété `fontFamily` ne doit contenir qu'un seul nom. Elle ne peut pas utiliser une liste de polices séparées par des virgules.
- Si vous utilisez des styles CSS pour les polices d'objets TextFields, donnez à la propriété CSS `font-family` le nom de la police incorporée. Si vous voulez utiliser une police incorporée, la propriété `font-family` ne doit contenir qu'un seul nom, et non pas une liste de noms.

Intégration d'une police dans Flash

L'outil de programmation Flash vous permet d'intégrer pratiquement toutes les polices installées sur votre système, notamment les polices TrueType et les polices Postscript Type 1.

Il existe plusieurs façons d'intégrer des polices dans une application. Vous pouvez par exemple :

- définir la police et les propriétés de style d'un objet TextField sur la Scène, puis en cocher la case Incorporer les polices ;
- créer et référencer un symbole de police ;
- créer et utiliser une bibliothèque d'exécution partagée contenant les symboles de la police intégrée.

Pour plus de détails sur l'intégration de polices dans les applications, consultez la section « Polices intégrées pour champs de texte dynamique ou de saisie » du guide *Utilisation de Flash*.

Contrôle de la netteté, de l'épaisseur et de l'anti-aliasing

Par défaut, Flash Player ou AIR détermine les paramètres de contrôle d'affichage du texte (netteté, épaisseur et lissage) qui s'appliquent lorsque le texte change de taille et de couleur ou s'affiche sur différents arrière-plans. Dans certains cas, vous pouvez définir ces paramètres, par exemple si le texte est très petit ou très gros, ou s'il s'affiche sur plusieurs arrière-plans. La classe `flash.text.TextRenderer` et les classes associées, telles que CSMSettings, permettent de remplacer les paramètres de Flash Player ou d'AIR. Elles offrent un contrôle précis de la qualité d'affichage du texte incorporé. Pour plus d'informations sur les polices intégrées, consultez la section « Utilisation de polices incorporées » à la page 456.

Remarque : *la propriété .antiAliasType de la classe `flash.text.TextField` doit avoir la valeur `AntiAliasType.ADVANCED` pour que vous puissiez définir la netteté, l'épaisseur ou la propriété gridFitType, ou pour que vous puissiez utiliser la méthode `TextRenderer.setAdvancedAntiAliasingTable()`.*

L'exemple suivant applique des propriétés personnalisées de modulation continue du trait (CSM) et de mise en forme au texte affiché, en utilisant la police incorporée `myFont`. Lorsque l'utilisateur clique sur le texte affiché, Flash Player ou Adobe AIR applique ces paramètres personnalisés :

```
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);
}
```

Utilisation du texte statique

Le texte statique peut uniquement être créé dans l'environnement de programmation Il est impossible d'instancier du texte statique en ActionScript. Le texte statique est utile si le contenu textuel est court et ne doit pas changer (contrairement au texte dynamique). Vous pouvez vous représenter le texte statique comme un élément graphique, comparable à un cercle ou un carré dessiné sur la Scène de l'outil de programmation Flash. Bien que le texte statique offre moins de possibilités que le texte dynamique, ActionScript 3.0 permet de lire les valeurs des propriétés du texte statique à l'aide de la classe StaticText. En outre, vous pouvez utiliser la classe TextSnapshot pour extraire des valeurs du texte statique.

Accès aux champs de texte statique à l'aide de la classe StaticText

On utilise en général la classe flash.text.StaticText dans le panneau Actions de Flash afin d'agir sur une occurrence de texte statique placée sur la scène. Vous pouvez également travailler dans des fichiers ActionScript qui interagissent avec le fichier SWF contenant le texte statique. Dans les deux cas, il est impossible de créer par code une occurrence de texte statique. Le texte est créé dans l'environnement de programmation.

Pour créer une référence à un champ de texte statique existant, vous pouvez effectuer une itération sur les éléments de la liste d'affichage et affecter une variable. Par exemple :

```
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);
}
}
```

Lorsque vous disposez d'une référence à un champ de texte statique, vous pouvez utiliser les propriétés de ce champ dans ActionScript 3.0. Le code suivant est associé à une image du scénario et suppose qu'une variable appelée `myFieldLabel` est affectée à une référence à un champ de texte statique. Un champ de texte dynamique `myField` est positionné relativement aux valeurs `x` et `y` de `myFieldLabel` et affiche à nouveau la valeur de `myFieldLabel`.

```
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
```

Utilisation de la classe TextSnapshot

Pour travailler par programmation avec une occurrence de texte statique existante, utilisez la classe flash.text.TextSnapshot pour modifier la propriété `textSnapshot` d'un objet flash.display.DisplayObjectContainer. En d'autres termes, créez une occurrence de TextSnapshot à partir de la propriété `DisplayObjectContainer.textSnapshot`. Vous pouvez alors appliquer des méthodes à cette occurrence afin d'extraire des valeurs ou de sélectionner des portions du texte statique.

Par exemple, placez un champ de texte statique contenant le texte « TextSnapshot Example » sur la scène. Ajoutez le code ActionScript suivant à l'image 1 du scénario :

```
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);
```

La classe TextSnapshot est utile pour extraire le texte des champs de texte statique dans un fichier SWF chargé, au cas où vous souhaiteriez utiliser ce texte comme valeur dans une autre zone de l'application.

Exemple TextField : mise en forme du texte dans le style « article de journal »

L'exemple « Article de journal » met en forme le texte pour lui donner l'aspect d'un article de journal. Le texte saisi peut contenir un gros titre, un intertitre et le corps de l'article. En fonction d'une largeur et d'une hauteur d'affichage, cet exemple met en forme le gros titre et l'intertitre pour qu'ils occupent toute la largeur disponible. Le texte de l'article est réparti sur plusieurs colonnes.

Cet exemple illustre les techniques de programmation en ActionScript suivantes :

- Extension de la classe TextField
- Chargement et application d'un fichier CSS externe
- Conversion de styles CSS en objets TextFormat
- Utilisation de la classe TextLineMetrics pour obtenir des informations sur la taille d'affichage du texte

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers de l'application News Layout se trouvent dans le dossier Samples/NewsLayout. L'application se compose des fichiers suivants :

Fichier	Description
NewsLayout.mxml ou NewsLayout.fla	Interface utilisateur de l'application pour Flex (MXML) ou Flash (FLA).
com/example/programmingas3/newslayout/StoryLayoutComponent.as	Classe Flex UIComponent qui place l'occurrence de StoryLayout.
com/example/programmingas3/newslayout/StoryLayout.as	Principale classe ActionScript chargée d'organiser les composants d'un article pour leur affichage.
com/example/programmingas3/newslayout/FormattedTextField.as	Sous-classe de la classe TextField qui gère son propre objet TextFormat.
com/example/programmingas3/newslayout/HeadlineTextField.as	Sous-classe de la classe FormattedTextField qui ajuste la taille des polices en fonction de la largeur voulue.
com/example/programmingas3/newslayout/MultiColumnTextField.as	Classe ActionScript qui répartit le texte de l'article sur plusieurs colonnes.
story.css	Fichier CSS définissant les styles du texte pour la mise en page.

Lecture du fichier CSS externe

L'application News Layout commence par récupérer le texte de l'article à partir d'un fichier XML local. Elle lit ensuite un fichier CSS externe contenant les informations de mise en forme pour le gros titre, l'intertitre et le texte.

Ce fichier CSS définit trois styles, un style de paragraphe standard pour l'article et les styles h1 et h2, respectivement pour le gros titre et l'intertitre.

```
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;
}
```

La technique utilisée pour lire le fichier CSS externe est identique à celle décrite à la section « Chargement de fichiers CSS externes » à la page 454. Après le chargement du fichier CSS, l'application exécute la méthode `onCSSFileLoaded()`, représentée ci-dessous.

```
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();
}
```

La méthode `onCSSFileLoaded()` crée un objet StyleSheet qui analyse les données du fichier CSS. Le texte principal de l'article est affiché dans un objet MultiColumnTextField, qui peut utiliser directement un objet StyleSheet. Toutefois, les champs du gros titre utilisent la classe HeadlineTextField, qui utilise un objet TextFormat pour sa mise en forme.

La méthode `onCSSFileLoaded()` appelle deux fois la méthode `getTextStyle()` pour convertir une déclaration de style CSS en un objet TextFormat destiné aux deux objets HeadlineTextField.

```
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;
}
```

Les noms de propriétés et la signification de leurs valeurs diffèrent entre les déclarations de style CSS et les objets TextFormat. La méthode `getTextStyle()` transforme donc les valeurs des propriétés CSS dans les valeurs attendues par l'objet TextFormat.

Disposition des éléments de l'article sur la page

La classe StoryLayout formate et met en page les champs de texte dévolus au gros titre, à l'intertitre et à l'article dans le style d'une page de journal. La méthode `displayText()` crée et place les divers champs.

```
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;
...
```

Chaque champ est placé sous le champ précédent en effectuant le calcul suivant : la propriété `y` du champ est égale à la propriété `y` du champ précédent, plus sa hauteur. Ce calcul dynamique de position est nécessaire, car les objets HeadlineTextField et MultiColumnTextField peuvent changer de hauteur en fonction de leur contenu.

Modification de la taille de la police en fonction de la taille du champ

Sur la base d'une largeur en pixels et d'un nombre maximum de lignes à afficher, l'objet HeadlineTextField modifie la taille de la police pour adapter le texte aux dimensions du champ. Si le texte est court, la police est de grande taille, créant ainsi un gros titre de style tabloïde. Si le texte est long, la police est de plus petite taille.

La méthode `HeadlineTextField.fitText()` reproduite ci-dessous est chargée du redimensionnement du texte :

```
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;
    }
}
```

La méthode `HeadlineTextField.fitText()` utilise une technique récursive simple pour dimensionner le texte. Elle estime d'abord un nombre moyen de pixels par caractère pour le texte, puis calcule une taille de départ. Elle change alors la taille de la police et vérifie si le texte a renvoyé des mots à la ligne et si le nombre maximal de lignes est dépassé. Si c'est le cas, elle appelle la méthode `shrinkText()` pour réduire la taille du texte, et teste à nouveau. Si le nombre de lignes n'est pas trop important, elle appelle la méthode `growText()` pour augmenter la taille du texte, et teste à nouveau. Ce processus s'interrompt lorsque l'augmentation de taille du texte d'un seul point crée un nombre de lignes trop élevé.

Répartition du texte sur plusieurs colonnes

La classe MultiColumnTextField répartit le texte entre plusieurs objets TextField qui sont organisés comme des colonnes de texte sur une page de journal.

Le constructeur de `MultiColumnTextField()` crée tout d'abord un tableau d'objets TextField, un pour chaque colonne :

```
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);
}
```

Chaque objet TextField est ajouté au tableau et à la liste d'affichage à l'aide de la méthode `addChild()`.

Si la propriété `text` ou `styleSheet` de StoryLayout change, la méthode `layoutColumns()` est appelée pour réafficher le texte. La méthode `layoutColumns()` appelle la méthode `getOptimalHeight()` pour déterminer la hauteur correcte en pixels nécessaire pour adapter tout le texte en fonction de la largeur disponible.

```
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;
    }
}
```

La méthode `getOptimalHeight()` calcule d'abord la largeur de chaque colonne. Elle définit ensuite la propriété `htmlText` du premier objet TextField du tableau. La méthode `getOptimalHeight()` utilise ce premier objet TextField pour connaître le nombre total de lignes renvoyées à la ligne, et en déduit donc le nombre de lignes optimal pour chaque colonne. Elle appelle ensuite la méthode `TextField.getLineMetrics()` pour obtenir un objet TextLineMetrics contenant des informations sur la taille du texte de la première ligne. La propriété `TextLineMetrics.height` représente la hauteur totale (en pixels) d'une ligne de texte, avec les mesures ascendantes, descendantes et l'interligne. La hauteur optimale de l'objet MultiColumnTextField est donc la hauteur de ligne multipliée par le nombre de lignes par colonne, plus 4 pour tenir compte de la bordure de deux pixels en haut et en bas de l'objet TextField.

Voici le code complet de la méthode `layoutColumns()` :

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

Lorsque la propriété `preferredHeight` a été définie par un appel à la méthode `getOptimalHeight`, la méthode `layoutColumns()` parcourt les objets TextField et définit la hauteur de chacun avec la valeur `preferredHeight`. La méthode `layoutColumns()` distribue ensuite le nombre de lignes de texte adapté à chaque champ, de sorte qu'aucun défilement ne se produise dans l'un d'eux et que le texte de chaque champ enchaîne sur celui du champ précédent. Si le style d'alignement du texte a été défini sur « justifier », la méthode `justifyLastLine()` est appelée pour justifier la ligne finale du texte dans un champ. Dans le cas contraire, la dernière ligne est considérée comme une ligne de fin de paragraphe et n'est donc pas justifiée.

Utilisation de Flash Text Engine

Flash Text Engine (FTE) fournit une faible prise en charge pour un contrôle sophistiqué des mesures de texte, de la mise en forme et du texte bidirectionnel. Il se caractérise par un flux de texte optimisé et une prise en charge des langues enrichie. Bien qu'il puisse être utilisé pour créer et gérer de simples éléments de texte, FTE est l'outil de base pour les développeurs qui souhaitent créer des composants d'édition de texte. En tant que tel, Flash Text Engine nécessite des connaissances avancées en programmation. Pour afficher de simples éléments de texte, consultez les sections précédentes qui décrivent l'utilisation de l'objet `TextField` et de ses objets associés.

Remarque : *Flash Text Engine est disponible à partir de Flash Player 10 et Adobe AIR 1.5.*

Création et affichage de texte

Les classes qui constituent Flash Text Engine vous permettent de créer, de mettre en forme et de contrôler le texte. Les classes suivantes constituent les éléments de base pour la création et l'affichage de texte avec Flash Text Engine :

- TextElement/GraphicElement/GroupElement : renferment le contenu d'une occurrence de TextBlock
- ElementFormat : spécifie les attributs de mise en forme du contenu d'une occurrence de TextBlock
- TextBlock : classe usine pour la création d'un paragraphe de texte
- TextLine : ligne de texte créée à partir de TextBlock

Pour afficher du texte, créez un objet TextElement à partir d'une chaîne, ainsi qu'un objet ElementFormat qui spécifie les caractéristiques de mise en forme, puis affectez l'objet TextElement à la propriété `content` d'un objet TextBlock. Créez les lignes de texte à afficher en appelant la méthode `TextBlock.createTextLine()`. Par exemple, le code suivant utilise ces classes FTE pour afficher « Hello World! This is Flash Text Engine! ».à l'aide du format et de la police par défaut.

```
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;
        }
    }
}
```

Les paramètres requis pour `createTextLine()` spécifient la ligne où doit commencer la nouvelle ligne et la largeur de celle-ci en pixels. La ligne où doit commencer la nouvelle ligne correspond généralement à la ligne précédente, mais dans le cas de la première ligne, la valeur est `null`.

Ajout d'objets GraphicElement et GroupElement

Vous pouvez affecter un objet GraphicElement à un objet TextBlock en vue d'afficher une image ou un élément graphique. Il vous suffit pour cela de créer une occurrence de la classe GraphicElement à partir d'un graphique ou d'une image, puis d'affecter l'occurrence à la propriété `TextBlock.content`. Créez la ligne de texte en appelant la méthode `TextBlock.createTextline()` selon la procédure habituelle. L'exemple suivant crée deux lignes de texte : une avec un objet GraphicElement, l'autre avec un objet TextElement.

```
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;
        }
    }
}
```

Vous pouvez créer un objet GroupElement afin de regrouper des objets TextElement, GraphicElement et d'autres objets GroupElement en vue de les affecter à la propriété `content` d'un objet TextBlock. Le paramètre au constructeur `GroupElement()` est un vecteur qui pointe vers le texte, le graphique et les éléments qui constituent le groupe. L'exemple suivant regroupe deux éléments graphiques et un élément de texte, et les affecte comme une unité à un bloc de texte.

```
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;
        }
    }
}
```

Remplacement du texte

Vous pouvez remplacer du texte dans une occurrence de TextBlock en appelant la méthode `TextElement.replaceText()` en vue de remplacer le texte dans l'objet TextElement que vous avez affecté à la propriété `TextBlock.content`. L'exemple suivant utilise `replaceText()` pour insérer du texte au début de la ligne, ajouter du texte à la fin de la ligne et remplacer du texte au milieu de la ligne.

```
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);
        }
    }
}
```

La méthode `replaceText()` remplace le texte spécifié à l'aide des paramètres `beginIndex` et `endIndex` par le texte spécifié à l'aide du paramètre `newText`. Si les valeurs des paramètres `beginIndex` et `endIndex` sont les mêmes, la méthode `replaceText()` insère le texte spécifié à cet emplacement. Dans le cas contraire, elle remplace les caractères spécifiés à l'aide des paramètres `beginIndex` et `endIndex` par le nouveau texte.

Gestion des événements dans FTE

Comme c'est le cas pour d'autres objets d'affichage, vous pouvez ajouter des écouteurs d'événement à une occurrence de TextLine. Par exemple, vous pouvez savoir à quel moment un utilisateur passe la souris sur une ligne de texte ou clique sur la ligne. L'exemple suivant détecte ces deux événements. Lorsque vous passez la souris sur une ligne, le curseur prend la forme d'un bouton et lorsque vous cliquez sur la ligne, il change de couleur.

```
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);
        }
    }
}
```

Copie miroir d'événements

Vous pouvez également matérialiser les événements d'un bloc de texte (ou d'une portion d'un bloc de texte) dans un diffuseur d'événements. Vous devez tout d'abord créer une occurrence de EventDispatcher, puis l'affecter à la propriété `eventMirror` d'une occurrence de TextElement. Si le bloc de texte n'est constitué que d'un seul élément de texte, le moteur de saisie effectue une copie miroir des événements pour l'intégralité du bloc de texte. Si le bloc de texte est constitué de plusieurs éléments de texte, le moteur de saisie effectue une copie miroir des événements uniquement pour les occurrences de TextElement dont la propriété `eventMirror` est définie. Dans l'exemple suivant, le texte est constitué de trois éléments : le mot « Click », le mot « here » et la chaîne « to see me in italic ». L'exemple affecte un diffuseur d'événements au deuxième élément de texte, le mot « here », et ajoute un écouteur d'événement, la méthode `clickHandler()`. La méthode `clickHandler()` met le texte en italique. Elle modifie également le contenu du troisième élément de texte et le remplace par la chaîne suivante : « 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);
        }
    }
}
```

Grâce aux fonctions `mouseOverHandler()` et `mouseOutHandler()`, le curseur prend la forme d'un bouton lorsqu'il est placé sur le mot « here » et reprend la forme d'une flèche lorsqu'il ne l'est pas.

Mise en forme du texte

Un objet `TextBlock` est un objet usine pour la création de lignes de texte. Le contenu d'un objet `TextBlock` est affecté via l'objet `TextElement`. Un objet `ElementFormat` gère la mise en forme du texte. La classe ElementFormat définit certaines propriétés, telles que l'alignement sur la ligne de base, le crénage, l'interlettrage, la rotation du texte, la taille et la couleur des polices, ainsi que la casse. Elle comprend également la méthode `FontDescription`, décrite en détail à la section « Utilisation des polices » à la page 478.

Utilisation de l'objet ElementFormat

Le constructeur de l'objet `ElementFormat` prend des nombreux paramètres facultatifs, dont `FontDescription`. Vous pouvez également définir ces propriétés en dehors du constructeur. L'exemple suivant illustre la relation des différents objets lors de la définition et de l'affichage d'une ligne de texte simple :

```
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);
        }
    }
}
```

Couleur de police et transparence (alpha)

La propriété `color` de l'objet `ElementFormat` définit la couleur de police. La valeur est un entier représentant les composants RVB de la couleur, par exemple : 0xFF0000 pour le rouge et 0x00FF00 pour le vert. La valeur par défaut est noir (0x000000).

La propriété `alpha` définit la valeur de transparence alpha d'un élément (`TextElement` et `GraphicElement`). La plage des valeurs est comprise entre 0 (complètement transparent) et 1 (complètement opaque), qui est la valeur par défaut. Les éléments dont la propriété `alpha` est de 0 sont invisibles, mais restent actifs. Cette valeur est multipliée par l'une des valeurs alpha héritées, ce qui rend l'élément plus transparent.

```
var ef:ElementFormat = new ElementFormat();
ef.alpha = 0.8;
ef.color = 0x999999;
```

Alignement et décalage de la ligne de base

La police et la taille du plus grand texte dans une ligne déterminent sa ligne de base dominante. Vous pouvez écraser ces valeurs en définissant `TextBlock.baselineFontDescription` et `TextBlock.baselineFontSize`. Vous pouvez aligner la ligne de base dominante sur l'une des lignes de bases du texte, à savoir la ligne ascendante, la ligne descendante, ou la ligne de base idéographique supérieure, centrale ou inférieure.

A. *Ascendante* ***B.*** *Ligne de base* ***C.*** *Descendante* ***D.*** *Hauteur-x*

Dans l'objet `ElementFormat`, trois propriétés déterminent la ligne de base et les caractéristiques d'alignement. La propriété `alignmentBaseline` définit la ligne de base principale d'un objet `TextElement` ou `GraphicElement`. Cette ligne de base est la ligne « d'accrochage ».de l'élément, et c'est à cette position que la ligne de base dominante du texte s'aligne.

La propriété `dominantBaseline` indique la ligne de base de l'élément à utiliser, qui détermine la position verticale de l'élément sur la ligne. La valeur par défaut est `TextBaseline.ROMAN`, mais vous pouvez également stipuler que la ligne de base `IDEOGRAPHIC_TOP` ou `IDEOGRAPHIC_BOTTOM` doit être dominante.

La propriété `baselineShift` déplace la ligne de base selon un nombre de pixels défini sur l'axe y. Dans un texte normal (aucune rotation), une valeur positive déplace la ligne de base vers le bas et une valeur négative la déplace vers le haut.

Casse typographique

La propriété `TypographicCase` de l'objet `ElementFormat` spécifie la casse du texte : majuscule, minuscule ou petites capitales.

```
var ef_Upper:ElementFormat = new ElementFormat();
ef_Upper.typographicCase = TypographicCase.UPPERCASE;

var ef_SmallCaps:ElementFormat = new ElementFormat();
ef_SmallCaps.typographicCase = TypographicCase.SMALL_CAPS;
```

Rotation du texte

Vous pouvez appliquer une rotation au bloc de texte ou aux glyphes d'un segment de texte par incréments de 90 degrés. La classe TextRotation définit les constantes suivantes pour définir la rotation du bloc de texte et des glyphes :

Constante	Valeur	Description
AUTO	“auto”	Spécifie une rotation vers la gauche de 90 degrés. Généralement utilisée avec un texte asiatique vertical pour faire pivoter uniquement les glyphes auxquelles il est nécessaire d'appliquer une rotation.
ROTATE_0	“rotate_0”	Spécifie aucune rotation.
ROTATE_180	“rotate_180”	Spécifie une rotation de 180 degrés.
ROTATE_270	“rotate_270”	Spécifie une rotation de 270 degrés.
ROTATE_90	“rotate_90”	Spécifie une rotation vers la droite de 90 degrés.

Pour appliquer une rotation aux lignes d'un texte, définissez tout d'abord la propriété `TextBlock.lineRotation` avant d'appeler la méthode `TextBlock.createTextLine()` pour créer la ligne de texte.

Pour appliquer une rotation aux glyphes dans un bloc de texte ou un segment, définissez la propriété `ElementFormat.textRotation` sur le nombre de degrés de rotation des glyphes souhaité. Une glyphe est la forme qui constitue un caractère, ou une partie d'un caractère qui consiste en plusieurs glyphes. La lettre a et le point sur un i, par exemple, sont des glyphes.

La rotation des glyphes est importante dans certaines langues asiatiques, notamment si vous souhaitez appliquer une rotation verticale aux lignes, mais ne pas appliquer de rotation aux caractères dans les lignes. Pour plus d'informations sur la rotation du texte asiatique, consultez la section « Justification du texte asiatique » à la page 482.

Voici un exemple de rotation du texte et des glyphes qu'il contient dans un texte asiatique : Cet exemple utilise également une police japonaise :

```
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);
        }
    }
}
```

Verrouillage et clonage d'un objet ElementFormat

Lorsqu'un objet `ElementFormat` est affecté à un type de `ContentElement`, sa propriété `locked` est automatiquement définie sur `true`. Toute tentative de modification d'un objet `ElementFormat` verrouillé renvoie une erreur `IllegalOperationError`. La meilleure pratique consiste à définir complètement ce type d'objet avant de l'affecter à l'occurrence de `TextElement`.

Si vous souhaitez modifier une occurrence de `ElementFormat` existante, vous devez tout d'abord vérifier sa propriété `locked`. Si elle est définie sur `true`, utilisez la méthode `clone()` pour créer une copie déverrouillée de l'objet. Vous pouvez modifier les propriétés de cet objet déverrouillé, puis l'affecter à l'occurrence de `TextElement`. Toute nouvelle ligne créée à partir de cet objet adopte la nouvelle mise en forme. Les lignes précédentes créées à partir de cet objet et utilisant l'ancienne mise en forme ne sont pas modifiées.

```
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);
        }
    }
}
```

Utilisation des polices

L'objet `FontDescription` est utilisé avec l'occurrence de `ElementFormat` pour identifier une police et définir certaines de ses caractéristiques. Ces caractéristiques incluent le nom de la police, le poids, la position, le rendu et la méthode de recherche de la police (police de périphérique/police intégrée).

Remarque : *FTE ne prend pas en charge les polices Type 1 ou les polices bitmap, telles que Type 3, ATC, CID ou CID basées sur SFNT.*

Définition des caractéristiques des polices (objet FontDescription)

La propriété `fontName` de l'objet `FontDescription` peut être un nom unique ou une liste de noms séparés par des virgules. Par exemple, dans une liste telle que « Arial, Helvetica, _sans », Flash Player recherche tout d'abord « Arial », puis « Helvetica » et finalement « _sans » s'il ne parvient pas à trouver les deux premières polices. La définition des noms de police comprend trois noms de police génériques : « _sans », « _serif ».et « _typewriter ». Ces noms correspondent à des polices de périphérique spécifiques, selon le système de lecture. Il est judicieux de spécifier ce type de noms par défaut dans toutes les descriptions de police qui utilisent des polices de périphérique. Si la propriété `fontName` n'est pas spécifiée, « _serif ».est utilisé comme nom par défaut.

La propriété `fontPosture` peut être définie sur la valeur par défaut (`FontPosture.NORMAL`) ou en italique (`FontPosture.ITALIC`). La propriété `fontWeight` peut être définie sur la valeur par défaut (`FontWeight.NORMAL`) ou en caractères gras (`FontWeight.BOLD`).

```
var fd1:FontDescription = new FontDescription();
fd1.fontName = "Arial, Helvetica, _sans";
fd1.fontPosture = FontPosture.NORMAL;
fd1.fontWeight = FontWeight.BOLD;
```

Polices intégrées ou polices de périphérique ?

La propriété `fontLookup` de l'objet `FontDescription` indique si Flash Player ou AIR recherche une police de périphérique ou une police intégrée pour rendre le texte. Si une police de périphérique (`FontLookup.DEVICE`) est spécifiée, le moteur d'exécution recherche la police sur le système de lecture. Si vous définissez une police intégrée (`FontLookup.EMBEDDED_CFF`), le moteur d'exécution recherche une police de ce type portant le nom indiqué dans le fichier SWF. Seules les polices CFF (Compact Font Format) intégrées utilisent ce paramètre. Si la police spécifiée est introuvable, une police de périphérique est utilisée.

Les polices de périphérique donnent lieu à des fichiers SWF moins volumineux. Les polices intégrées garantissent une plus grande homogénéité d'une plate-forme à l'autre

```
var fd1:FontDescription = new FontDescription();
fd1.fontLookup = FontLookup.EMBEDDED_CFF;
fd1.fontName = "Garamond, _serif";
```

Mode de rendu et repères

Le rendu CFF (Compact Font Format) est disponible à partir de Flash Player 10 et Adobe AIR 1.5. Ce type de rendu de police permet une meilleure lisibilité du texte et un affichage optimisé des caractères de petite taille. Ce paramètre s'applique uniquement aux polices intégrées. La valeur par défaut de `FontDescription` correspond à ce paramètre (`RenderingMode.CFF`) pour la propriété `renderingMode`. Vous pouvez définir cette propriété sur `RenderingMode.NORMAL` afin qu'elle corresponde au type de rendu utilisé par Flash Player 7 ou versions antérieures.

Lorsque le rendu CFF est sélectionné, une deuxième propriété, `cffHinting`, permet de contrôler la manière dont les corps horizontaux d'une police sont adaptés à la grille de sous-pixels. La valeur par défaut, `CFFHinting.HORIZONTAL_STEM`, utilise les repères CFF. Si vous définissez cette propriété sur `CFFHinting.NONE` , les repères sont supprimés. Ce paramètre convient pour les animations ou les grandes tailles de police.

```
var fd1:FontDescription = new FontDescription();
fd1.renderingMode = RenderingMode.CFF;
fd1.cffHinting = CFFHinting.HORIZONTAL_STEM;
```

Verrouillage et clonage d'un objet FontDescription

Lorsque vous affectez une occurrence de `ElementFormat` à un objet `FontDescription`, la propriété `locked` de ce dernier est automatiquement définie sur `true`. Toute tentative de modification d'un objet `FontDescription` verrouillé renvoie une erreur `IllegalOperationError`. La meilleure pratique consiste à définir complètement ce type d'objet avant de l'affecter à l'occurrence de `ElementFormat`.

Si vous souhaitez modifier un objet `FontDescription` existant, vous devez tout d'abord vérifier sa propriété `locked`. Si elle est définie sur `true`, utilisez la méthode `clone()` pour créer une copie déverrouillée de l'objet. Vous pouvez modifier les propriétés de cet objet déverrouillé, puis l'affecter à `ElementFormat`. Toute nouvelle ligne créée à partir de ce `TextElement` adopte la nouvelle mise en forme. Les lignes précédentes créées à partir de ce même objet restent inchangées.

```
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);
        }
    }
}
```

Contrôle du texte

FTE vous propose un ensemble de commandes de mise en forme du texte afin de gérer la justification et l'espacement des caractères (crénage et interlettrage). Il existe également des propriétés permettant de détecter les lignes brisées et de définir des taquets de tabulation dans les lignes.

Justification du texte

En ajustant l'espacement entre les mots, et parfois entre les lettres, toutes les lignes d'un paragraphe possèdent une longueur identique. Bien que l'espacement entre les mots et les lettres varie, le texte est aligné des deux côtés. Les colonnes de texte dans les journaux et les magazines sont le plus souvent justifiées.

La propriété `lineJustfication` de la classe SpaceJustifier vous permet de contrôler la justification des lignes dans un bloc de texte. La classe LineJustification définit des constantes en vue de spécifier une option de justification : `ALL_BUT_LAST` justifie tout le texte, à l'exception de la dernière ligne ; `ALL_INCLUDING_LAST` justifie tout le texte, y compris la dernière ligne ; l'option par défaut, `UNJUSTIFIED`, ne justifie pas le texte.

Pour justifier le texte, définissez la propriété `lineJustification` sur une occurrence de la classe SpaceJustifier, puis affectez celle-ci à la propriété `textJustifier` d'une occurrence de TextBlock. L'exemple suivant crée un paragraphe dans lequel la totalité du texte est justifiée, à l'exception de la dernière ligne.

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

Pour varier l'espacement entre les lettres et entre les mots, définissez la propriété `SpaceJustifier.letterspacing` sur `true`. L'activation de l'espacement entre les lettres peut réduire le nombre d'espaces disgracieux entre les mots, qui se produisent parfois lors d'une simple justification.

Justification du texte asiatique

Pour justifier un texte asiatique, il faut tenir compte d'autres considérations. Le texte peut être écrit de haut en haut et certains caractères appelés kinsoku ne peuvent pas apparaître au début ou à la fin d'une ligne. La classe JustificationStyle définit les constantes suivantes, qui spécifient les options de gestion de ces caractères. `PRIORITIZE_LEAST_ADJUSTMENT` base la justification sur l'expansion ou sur la compression de la ligne, en fonction de celle qui produit les meilleurs résultats. `PUSH_IN_KINSOKU` base la justification sur la compression des caractères kinsoku à la fin de la ligne, ou sur l'expansion de la ligne s'il n'existe aucun caractère kinsoku ou si cet espace est insuffisant.

`PUSH_OUT_ONLY` base la justification sur l'expansion de la ligne. Pour créer un bloc de texte asiatique vertical, définissez la propriété `TextBlock.lineRotation` sur `TextRotation.ROTATE_90`, puis définissez la propriété `ElementFormat.textRotation` sur `TextRotation.AUTO` (paramètre par défaut). Si vous définissez la propriété `textRotation` sur `AUTO`, les glyphes dans le texte restent verticales au lieu de pivoter sur le côté lors de la rotation de la ligne. Le paramètre `AUTO` effectue une rotation vers la gauche de 90 degrés pour les glyphes complètes uniquement, comme le spécifient les propriétés Unicode de la glyphe. L'exemple suivant affiche un bloc de texte japonais vertical et le justifie à l'aide de l'option `PUSH_IN_KINSOKU`. i

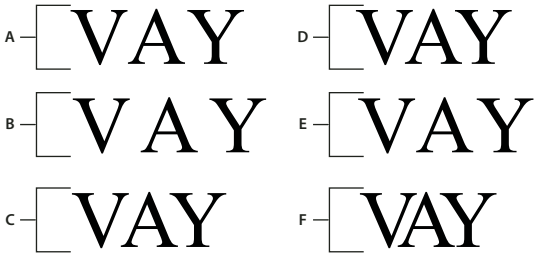
```
package
{
    import flash.text.engine.*;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.system.Capabilities;

    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;
        }
    }
  }
}
```

Crénage et interlettrage

Le crénage et l'interlettrage influent sur la distance entre les paires de caractères adjacentes dans un bloc de texte. Le crénage contrôle la manière dont les paires de caractères s'assemblent, notamment les paires « WA ».ou « Va ». Il est défini dans l'objet `ElementFormat`. Par défaut, cet effet est activé (`Kerning.ON`) ; il peut être réglé sur OFF ou AUTO, auquel cas le crénage n'est appliqué qu'entre les caractères autres que Kanji, Hiragana ou Katakana.

L'interlettrage ajoute ou soustrait un nombre de pixels défini entre les caractères dans un bloc de texte ; cet effet est également défini dans l'objet `ElementFormat`. L'interlettrage fonctionne avec les polices intégrées et les polices de périphérique. FTE prend en charge deux propriétés d'interlettrage : `trackingLeft`, qui ajoute ou soustrait des pixels à gauche d'un caractère, et `trackingRight`, qui ajoute ou soustrait des pixels à droite d'un caractère. Si vous utilisez le crénage, la valeur d'interlettrage est ajoutée aux valeurs de crénage ou soustraite des valeurs de crénage de chaque paire de caractères.



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;
```

Texte avec retour à la ligne automatique

La propriété `breakOpportunity` de l'objet `ElementFormat` indique les caractères pouvant être utilisés pour le renvoi lorsque le texte est renvoyé sur plusieurs lignes. La valeur par défaut, `BreakOpportunity.AUTO`, utilise les propriétés Unicode standard, telles que le saut entre les mots et sur les traits d'union. Le paramètre `BreakOpportunity.ALL` permet de considérer un caractère comme une opportunité de saut de ligne, ce qui est très utile lors de la création d'effets, tels que le texte le long d'un tracé.

```
var ef:ElementFormat = new ElementFormat();
ef.breakOpportunity = BreakOpportunity.ALL;
```

Taquets de tabulation

Pour définir des taquets de tabulation dans un bloc de texte, définissez les taquets de tabulation en créant des occurrences de la classe TabStop. Les paramètres du constructeur `TabStop()` indiquent la méthode d'alignement du texte avec le taquet de tabulation. Ces paramètres indiquent la position du taquet de tabulation, et pour l'alignement décimal, la valeur d'alignement, exprimée sous forme de chaîne. En général, cette valeur est un point décimal, mais elle pourrait également être une virgule, ou le symbole du dollar, du yen ou de l'euro, entre autres. La ligne de code suivante crée un taquet de tabulation appelé tab1.

```
var tab1:TabStop = new TabStop(TabAlignment.DECIMAL, 50, ".");
```

Après avoir créé les taquets de tabulation d'un bloc de texte, affectez-les à la propriété `tabStops` d'une occurrence de TextBlock. Etant donné que la propriété `tabStops` nécessite un vecteur, créez tout d'abord un vecteur, puis ajoutez la tabulation pour l'arrêter. Le vecteur vous permet d'affecter un ensemble de taquets de tabulation au bloc de texte. L'exemple suivant crée une occurrence de `Vector<TabStop>` et lui ajoute un ensemble d'objets TabStop. Affectez ensuite les taquets de tabulation à la propriété `tabStops` d'une occurrence de TextBlock.

```
var tabStops:Vector.<TabStop> = new Vector.<TabStop>();
tabStops.push(tab1, tab2, tab3, tab4);
textBlock.tabStops = tabStops
```

Pour plus d'informations sur les vecteurs, consultez la section « Utilisation de tableaux » à la page 158

L'exemple suivant illustre l'effet de chacune des options d'alignement de l'objet TabStop.

```
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 FontDdscription("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;
            }
        }
    }
}
```

Exemple FTE : article de journal

Cet exemple de code illustre l'utilisation de FTE (Flash Text Engine) pour mettre en forme une page de journal simple. Cette page comprend un gros titre, un sous-titre et une section sur plusieurs colonnes.

Créez d'abord un fichier FLA et rattachez le code suivant au cadre n° 2 de la couche par défaut :

```
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();
```

Dans cet exemple, StoryLayout.as est le script contrôleur. Il définit le contenu, lit les informations de style issues d'une feuille de style externe et les affecte aux objets ElementFormat. Il crée ensuite le gros titre, le sous-titre et les éléments de texte sur plusieurs colonnes.

```
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 est utilisé en tant que classe de base pour la création des blocs de texte. Il comprend également des fonctions permettant de modifier la taille de police et la casse.

```
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 étend la classe FormattedTextBlock et est utilisé pour créer les gros titres. Il contient une fonction destinée à faire tenir le texte dans un espace déterminé sur la page.

```
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 gère la mise en forme du texte dans un format multicolonne. Il illustre la souplesse d'un objet TextBlock, utilisé comme usine pour créer, mettre en forme et positionner les lignes de texte.

```
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);
        }

    }
}
```

Chapitre 22 : Utilisation des images bitmap

Outre ses possibilités en matière de dessin vectoriel, ActionScript 3.0 permet également de créer des images bitmap ou de manipuler les données de pixels d'images bitmap externes chargées dans un fichier SWF. Cette possibilité de lire et modifier des valeurs de pixel individuelles permet de créer des effets visuels comme ceux des filtres et d'utiliser les fonctions intégrées de gestion du « bruit » pour créer des textures et des bruits aléatoires. Toutes ces techniques sont décrites dans le présent chapitre.

Principes de base de l'utilisation des images bitmap

Introduction à l'utilisation des images bitmap

Tout travail avec des images numériques nécessite de gérer deux types de graphismes : les bitmaps et les graphismes vectoriels. Les images bitmap, également appelées graphismes en points, sont composées de petits carrés (les pixels) organisés en une grille rectangulaire. De leur côté, les graphismes vectoriels sont composés de formes géométriques (lignes, courbes et polygones) générées à l'aide de formules mathématiques.

Les images bitmap sont définies par la largeur et la hauteur de l'image, mesurées en pixels, et par le nombre de bits contenu dans chaque pixel (ce nombre de bits définit le nombre de couleurs que peut comporter l'image). Dans le cas d'une image bitmap utilisant le modèle colorimétrique RVB, les pixels sont composés de trois octets : rouge, vert et bleu. Chaque octet contient une valeur comprise entre 0 et 255. C'est la combinaison de ces octets pour chaque pixel qui produit une couleur, un peu comme le mélange des couleurs de base par un peintre. Par exemple, un pixel contenant les valeurs 255, 102 et 0 respectivement pour les octets dévolus au rouge, au vert et au bleu produira un orange vif.

La qualité d'une image bitmap est déterminée en combinant la résolution en pixels de l'image avec sa profondeur de couleur exprimée en bits ou en octets. La *résolution* définit le nombre de pixels contenus dans l'image. Plus le nombre de pixels est important, plus la résolution est élevée et plus l'image semble bien définie. La *profondeur de couleurs* définit la quantité d'informations colorimétriques contenues par chaque pixel. Par exemple, une image ayant une profondeur de couleur de 16 bits par pixel ne peut pas représenter un nombre de nuances aussi élevé qu'une image ayant une profondeur de couleur de 48 bits. En conséquence, l'image sur 48 bits aura plus de nuances que la version sur 16 bits.

Dans la mesure où les images bitmap dépendent de la résolution, il est délicat de modifier leur échelle, ce qui se remarque particulièrement avec les images bitmap agrandies, qui perdent beaucoup de détails et de qualité.

Format des fichiers bitmap

Les images bitmap existent en divers formats de fichier. Ces formats utilisent différents types d'algorithmes de compression pour réduire la taille des fichiers et optimiser la qualité de l'image en fonction de sa destination. BMP, GIF, JPG, PNG et TIFF sont les formats d'image bitmap pris en charge par Adobe Flash Player et Adobe AIR.

BMP

Le format BMP (pixellisé) est un format d'image par défaut utilisé par le système d'exploitation Microsoft Windows. Il ne fait appel à aucune forme d'algorithme de compression ; il en résulte généralement des tailles de fichiers assez importantes.

GIF

Le format GIF (Graphics Interchange Format) fut développé à l'origine par CompuServe en 1987, dans le but de transmettre des images en 256 couleurs (codées sur 8 bits). Ce format, qui permet d'obtenir des fichiers de petite taille, est très utilisé pour les images sur le Web. Toutefois, en raison de son nombre limité de couleurs, ce format n'est pas adapté aux photographies, qui nécessitent en général un nombre de nuances plus élevé. Les images GIF autorisent la transparence sur un bit, ce qui permet de rendre les couleurs invisibles (ou transparentes). C'est ainsi que sont produits les fonds transparents des images de pages Web.

JPEG

Développé par le Joint Photographic Experts Group (JPEG), le format d'image JPEG (souvent noté JPG) fait appel à un algorithme de compression avec perte pour autoriser une profondeur de couleur de 24 bits avec une taille de fichier très réduite. En raison de la compression avec perte, chaque fois que l'image est enregistrée elle perd des données, donc de la qualité, mais la taille de fichier résultante en est d'autant plus réduite. Le format JPEG est idéal pour les photographies, car il permet d'afficher des millions de couleurs différentes. La possibilité de contrôler le taux de compression appliqué à l'image permet de modifier la qualité de l'image et la taille du fichier.

PNG

Le format PNG (Portable Network Graphics) a été créé comme autre possibilité gratuite (open source) au format de fichier GIF, qui est breveté. Les fichiers PNG acceptent jusqu'à 64 bits de profondeur de couleur, ce qui permet d'obtenir jusqu'à 16 millions de couleurs. Le format PNG étant relativement récent, les versions anciennes de certains navigateurs ne gèrent pas ces fichiers. Contrairement au JPG, le format PNG utilise une compression sans perte, ce qui signifie qu'aucune donnée de l'image n'est perdue lors de l'enregistrement. De plus, les fichiers PNG acceptent également la transparence alpha, ce qui autorise jusqu'à 256 niveaux de transparence.

TIFF

Le format TIFF (ou Tagged Image File Format) était le format multiplate-forme de choix avant l'arrivée de PNG. Le format TIFF a un inconvénient qui est la multiplicité de ses variétés : aucun lecteur n'est en mesure de traiter toutes les versions disponibles. De surcroît, les navigateurs Web ne prennent pas en charge le format actuellement. TIFF peut utiliser une compression avec ou sans pertes et il est mesure de traiter des espaces de couleurs spécifiques au périphérique tels que CMJN (cyan-magenta-jaune-noir).

Fichiers bitmap transparents et opaques

Les images bitmap qui utilisent le format GIF ou PNG peuvent comporter pour chaque pixel un octet supplémentaire pour le canal alpha. Cet octet de pixel supplémentaire représente la valeur de transparence du pixel.

Les images GIF n'autorisent la transparence que sur la base d'un bit, ce qui ne permet de spécifier la transparence que pour une seule couleur (dans une palette de 256 couleurs). Par contre, les images PNG acceptent jusqu'à 256 niveaux de transparence. Cette caractéristique est particulièrement intéressante pour « fondre » les images ou le texte avec l'arrière-plan.

ActionScript 3.0 permet de gérer cet octet supplémentaire de transparence à l'aide de la classe BitmapData. Suivant le modèle de transparence PNG, la constante `BitmapDataChannel.ALPHA` permet d'obtenir jusqu'à 256 niveaux de transparence.

Tâches courantes d'utilisation des images bitmap

Voici plusieurs tâches courantes susceptibles d'être accomplies en cas d'utilisation d'images bitmap en ActionScript:

- Affichage de bitmaps
- Récupération et modification des valeurs de couleur des pixels

- Copie de données bitmap :
  - Création d'une copie exacte d'un bitmap
  - Copie des données d'un canal couleur d'une image bitmap dans un canal couleur d'une autre image bitmap
  - Copie sous forme de bitmap d'un objet d'affichage apparaissant à l'écran
- Création de bruit et de textures sous forme d'images bitmap
- Défilement du contenu d'images bitmap

Concepts importants et terminologie

La liste suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Alpha : niveau de transparence (ou, plus précisément, d'opacité) d'une couleur ou d'une image. La valeur alpha est fréquemment appelée valeur du *canal alpha*.
- Couleur ARVB : modèle colorimétrique suivant lequel la couleur de chaque pixel est définie selon ses composants rouge, vert et bleu, ainsi que par sa transparence spécifiée comme valeur alpha.
- Canal de couleur : en général, les couleurs sont représentées par le mélange des couleurs primaires, rouge, vert et bleu (pour les graphiques sur ordinateur). Chaque couleur primaire est considérée comme un canal de couleur. C'est le mélange des proportions de chaque canal de couleur qui détermine la couleur finale.
- Profondeur de couleur : parfois appelée *codage des couleurs*, cette caractéristique détermine la quantité de mémoire vive dévolue à chaque pixel, ce qui, indirectement, définit le nombre de couleurs qu'il est possible d'afficher.
- Pixel : unité d'information de base pour une image bitmap (en essence, un point coloré).
- Résolution : dimensions d'une image en pixels, ce qui détermine le niveau de détails fins qu'il est possible de séparer dans l'image. La résolution est fréquemment exprimée en nombre de pixels pour la largeur et la hauteur.
- Couleur RVB : modèle colorimétrique suivant lequel la couleur de chaque pixel est définie selon ses composants rouge, vert et bleu.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester ses exemples de code. Etant donné que ce chapitre décrit comment créer et manipuler du contenu visuel, le test du code implique l'exécution du code et l'affichage des résultats dans le fichier SWF créé.

Pour tester les exemples de code de ce chapitre :

1. Créez un document vide à l'aide de l'outil de programmation Flash.
2. Sélectionnez une image-clé dans le scénario.
3. Ouvrez le panneau Actions et copiez le code dans le panneau Script.
4. Exécutez le programme en sélectionnant Contrôle >Tester l'animation.

   Les résultats du code apparaissent dans le fichier SWF créé.

Presque tous les exemples de code créent une image bitmap. Vous pouvez donc les tester directement sans avoir à fournir une image bitmap. Si vous souhaitez tester le code sur votre propre image, vous pouvez importer celle-ci dans Adobe Flash CS4 Professional ou la charger dans le SWF de test et utiliser les données bitmap avec l'exemple de code. Pour plus d'informations sur le chargement d'images externes, consultez la section « Chargement dynamique du contenu d'affichage » à la page 319.

Classes Bitmap et BitmapData

ActionScript 3.0 propose deux classes principales qu'il est possible d'utiliser avec les images bitmap : la classe Bitmap, qui est utilisée pour afficher des images bitmap à l'écran ; puis la classe BitmapData, qui est utilisée pour accéder aux données d'image brutes d'un bitmap et les manipuler.

Présentation de la classe Bitmap

Sous-classe de la classe DisplayObject, la classe Bitmap est la principale classe utilisée en ActionScript 3.0 pour l'affichage d'images bitmap. Ces images peuvent avoir été chargées dans Flash Player ou Adobe AIR via la classe flash.display.Loader, ou créées dynamiquement à l'aide du constructeur `Bitmap()`. En cas de chargement d'une image provenant d'une source externe, un objet Bitmap ne peut contenir que des images aux formats GIF, JPEG ou PNG. L'occurrence de l'objet Bitmap peut être considérée comme enveloppe d'un objet BitmapData devant être affiché sur la scène. Une occurrence de Bitmap étant un objet d'affichage, toutes les caractéristiques et fonctionnalités des objets d'affichage peuvent être utilisées pour la manipuler. Pour plus d'informations sur l'utilisation des objets d'affichage, consultez le chapitre « Programmation de l'affichage » à la page 277.

Accrochage et lissage des pixels

Outre les fonctionnalités communes à tous les objets d'affichage, la classe Bitmap dispose de quelques fonctionnalités supplémentaires, propres aux images bitmap.

La propriété `pixelSnapping` de la classe Bitmap, dont l'effet est similaire à la fonction d'accrochage des pixels de l'environnement Flash, détermine si un objet Bitmap doit être « aimanté » par le pixel le plus proche. Cette propriété accepte l'une des trois constantes définies dans la classe PixelSnapping : `ALWAYS`, `AUTO` et `NEVER`.

La syntaxe de l'accrochage aux pixels est la suivante :

```
myBitmap.pixelSnapping = PixelSnapping.ALWAYS;
```

Lorsque des images bitmap sont redimensionnées, il est fréquent qu'elles deviennent floues et distordues. Pour réduire cette distorsion, utilisez la propriété `smoothing` de la classe BitmapData. Lorsque cette propriété booléenne a la valeur `true`, elle adoucit les pixels par un effet d'anti-aliasage appliqué aux images redimensionnées. Celles-ci ont alors un aspect plus clair, plus naturel.

Présentation de la classe BitmapData

La classe BitmapData, qui se trouve dans le package flash.display, peut être comparée à un cliché photographique des pixels d'une image bitmap, que celle-ci soit chargée ou créée dynamiquement. Ce cliché est représenté par une grille de données des pixels de l'objet. La classe BitmapData contient également une série de méthodes permettant de créer et modifier des données de pixels.

Pour instancier un objet BitmapData, utilisez le code suivant :

```
var myBitmap:BitmapData = new BitmapData(width:Number, height:Number, transparent:Boolean,
fillColor:uinit);
```

Les paramètres `width` et `height` permettent d'indiquer la largeur et la hauteur du bitmap. Ils acceptent une valeur maximum de 2880 pixels. Le paramètre `transparent` indique si l'image bitmap comporte un canal alpha (`true`) ou non (`false`). Le paramètre `fillColor` est une valeur colorimétrique sur 32 bits qui spécifie la couleur d'arrière-plan, ainsi que la valeur de la transparence (si cette dernière est `true`). L'exemple suivant crée un objet BitmapData dont l'arrière-plan orange est transparent à 50 % :

```
var myBitmap:BitmapData = new BitmapData(150, 150, true, 0x80FF3300);
```

Pour afficher un objet BitmapData nouvellement créé, affectez-le à une occurrence de Bitmap. Pour ce faire, vous pouvez soit transmettre l'objet BitmapData sous forme de paramètre du constructeur de l'objet Bitmap, ou l'affecter à la propriété `bitmapData` d'une occurrence de Bitmap existante. Vous devez également affecter cette occurrence de Bitmap à la liste d'affichage en appelant la méthode `addChild()` ou `addChildAt()` du conteneur d'objet d'affichage qui contiendra cette occurrence de Bitmap. Pour plus d'informations sur l'utilisation de la liste d'affichage, consultez la section « Ajout d'objets d'affichage à la liste d'affichage » à la page 285.

L'exemple suivant crée un objet BitmapData avec un remplissage rouge, et l'affiche dans une occurrence de Bitmap :

```
var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false, 0xFF0000);
var myImage:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myImage);
```

Manipulation des pixels

La classe BitmapData contient des méthodes qui permettent de modifier les valeurs des données de pixels.

Manipulation individuelle de pixels

Pour modifier une image bitmap au niveau des pixels, il est d'abord nécessaire d'obtenir les valeurs colorimétriques des pixels de la zone à modifier. La méthode `getPixel()` permet d'obtenir ces valeurs de pixels.

La méthode `getPixel()` renvoie les valeurs RVB des coordonnées (de pixels) x et y qui lui sont passées en paramètres. Si l'un des pixels comporte des informations de transparence (canal alpha), il est nécessaire d'utiliser la méthode `getPixel32()`. Cette méthode récupère également une valeur RVB, mais contrairement à ce qui se passe avec `getPixel()`, la valeur renvoyée par `getPixel32()` contient des données supplémentaires qui représentent la valeur du canal alpha (transparence) du pixel sélectionné.

Pour simplement modifier la couleur ou la transparence d'un pixel contenu dans un bitmap, il est aussi possible d'utiliser la méthode `setPixel()` ou `setPixel32()`. Pour définir la couleur d'un pixel, il suffit de passer les coordonnées x et y et la valeur colorimétrique à l'une de ces méthodes.

Dans l'exemple suivant, `setPixel()` est utilisée pour tracer une croix sur un fond vert BitmapData. La méthode `getPixel()` permet ensuite de récupérer la valeur colorimétrique du pixel ayant les coordonnées 50, 50 et de suivre la valeur renvoyée.

```
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));
```

Pour lire la valeur d'un groupe de pixels, et non pas d'un pixel isolé, utilisez la méthode `getPixels()`. Cette méthode génère un tableau d'octets à partir d'une zone rectangulaire de données de pixels, et le passe en paramètre. Chaque élément du tableau d'octets (autrement dit, les valeurs des pixels) est un entier non signé (valeurs non multipliées sur 32 bits).

Inversement, pour modifier (ou définir) la valeur d'un groupe de pixels, utilisez la méthode `setPixels()`. Cette méthode attend deux paramètres (`rect` et `inputByteArray`), qui sont combinés pour produire une zone rectangulaire (`rect`) de données de pixels (`inputByteArray`).

Au fur et à mesure de la lecture (ou de l'écriture) des données dans `inputByteArray`, la méthode `ByteArray.readUnsignedInt()` est appelée pour chaque pixel du tableau. Si pour une raison quelconque `inputByteArray` ne contient pas un rectangle complet de données de pixels, la méthode interrompt le traitement des données de l'image.

Il est important de comprendre que pour lire ou modifier des données de pixels, le tableau d'octets attend les valeurs suivantes sur 32 octets : alpha, rouge, vert, bleu (ARVB).

L'exemple suivant utilise les méthodes `getPixels()` et `setPixels()` pour copier un groupe de pixels d'un objet BitmapData à un autre :

```
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;
```

Détection de collision au niveau des pixels

La méthode `BitmapData.hitTest()` effectue une détection de collision au niveau des pixels entre les données bitmap et celles d'un autre objet ou d'un point.

La méthode `BitmapData.hitTest()` accepte cinq paramètres :

- `firstPoint` (Point) : ce paramètre référence la position du coin supérieur gauche du premier objet BitmapData sur lequel le test de collision doit être effectué.
- `firstAlphaThreshold` (uint) : ce paramètre spécifie la valeur la plus élevée du canal alpha considéré comme étant opaque pour ce test de collision.
- `secondObject` (Object) : ce paramètre représente la zone d'impact. L'objet `secondObject` peut être un objet de type Rectangle, Point, Bitmap ou BitmapData. Cet objet représente la zone dans laquelle doit être effectuée la détection de collision.

- `secondBitmapDataPoint` (Point) : ce paramètre facultatif définit l'emplacement d'un pixel dans le deuxième objet BitmapData. Utilisez uniquement ce paramètre lorsque la valeur de `secondObject` est un objet BitmapData. La valeur par défaut est `null`.
- `secondAlphaThreshold` (uint) : ce paramètre facultatif représente la valeur la plus élevée de canal alpha considérée comme étant opaque dans le deuxième objet BitmapData. La valeur par défaut est 1. Utilisez uniquement ce paramètre lorsque la valeur de `secondObject` est un objet BitmapData et que les deux objets BitmapData sont transparents.

Lors d'une détection de collision sur des images opaques, n'oubliez pas qu'ActionScript traite l'image comme si c'était un rectangle entièrement opaque (ou un cadre de sélection). Par ailleurs, lors de tests de collision au niveau des pixels pour des images qui sont transparentes, les deux images doivent être transparentes. De plus, ActionScript utilise les paramètres de seuil alpha pour déterminer le point auquel les pixels passent de transparents à opaques.

L'exemple suivant crée trois images bitmap et teste une éventuelle collision de pixels en deux différents points (l'un renvoie false, l'autre 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
```

Copie de données bitmap

Pour copier des données bitmap d'une image à une autre, vous disposez de plusieurs méthodes : `clone()`, `copyPixels()`, `copyChannel()` et `draw()`.

Comme son nom l'indique, la méthode `clone()` permet de cloner, ou échantillonner, des données bitmap d'un objet BitmapData à un autre. Cette méthode renvoie un nouvel objet BitmapData qui est un clone exact de l'occurrence originale.

L'exemple suivant clone une copie d'un carré orange (parent) et place le clone à côté du carré parent original :

```
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;
```

La méthode `copyPixels()` permet de copier rapidement et aisément des pixels d'un objet BitmapData dans un autre. Cette méthode prend un « cliché » rectangulaire (défini par le paramètre `sourceRect`) de l'image source et le copie dans une autre zone rectangulaire de taille égale. L'emplacement du rectangle ainsi « collé » est défini par le paramètre `destPoint`.

La méthode `copyChannel()` analyse une valeur de canal de couleur prédéfini (alpha, rouge, vert ou bleu) dans un objet source BitmapData et la copie dans un canal donné de l'objet BitmapData de destination. Cette méthode n'affecte pas les autres canaux de l'objet BitmapData de destination.

La méthode `draw()` dessine, ou affiche, le contenu graphique d'un objet d'affichage source (Sprite, Clip, etc.) dans un nouveau bitmap. Les paramètres `matrix`, `colorTransform`, `blendMode` et `clipRect` permettent de modifier l'aspect du nouveau bitmap. Cette méthode utilise le programme de rendu vectoriel de Flash Player et AIR pour générer les données.

Pour appeler la méthode `draw()`, vous devez lui transmettre l'objet source (Sprite, Clip ou tout autre objet d'affichage.) comme premier paramètre, comme ci-dessous :

```
myBitmap.draw(movieClip);
```

Si des transformations (couleur, matrice, etc.) ont été appliquées à l'objet source après son chargement, ces transformations ne sont pas copiées dans le nouvel objet. Pour copier les transformations dans le nouveau bitmap, vous devez copier la valeur de la propriété `transform` de l'objet original dans la propriété `transform` de l'objet Bitmap qui utilise le nouvel objet BitmapData.

Création de textures avec les fonctions de bruit aléatoire

Pour modifier l'aspect d'un bitmap, vous pouvez lui appliquer un effet de bruit à l'aide de la méthode `noise()` ou de la méthode `perlinNoise()`. La « neige » qui apparaît sur l'écran d'un téléviseur mal réglé est du bruit, ou du souffle.

Pour appliquer un effet de bruit à un bitmap, utilisez la méthode `noise()`. Cette méthode applique une valeur colorimétrique aléatoire aux pixels de la zone spécifiée d'une image bitmap.

Cette méthode accepte cinq paramètres :

- `randomSeed` (int) : valeur aléatoire de départ qui déterminera le motif. En dépit du nom de ce paramètre, une même valeur pour ce nombre produira toujours le même résultat. Pour obtenir un résultat véritablement aléatoire, utilisez la méthode `Math.random()` pour transmettre un nombre aléatoire pour ce paramètre.
- `low` (uint) : ce paramètre représente la valeur la plus faible à générer pour chaque pixel (de 0 à 255). Sa valeur par défaut est 0. Les valeurs les plus basses produisent un motif de bruit sombre, les valeurs les plus élevées produisent un motif de plus en plus clair.
- `high` (uint) : ce paramètre représente la valeur la plus élevée à générer pour chaque pixel (de 0 à 255). La valeur par défaut est 255. Les valeurs les plus basses produisent un motif de bruit sombre, les valeurs les plus élevées produisent un motif de plus en plus clair.
- `channelOptions` (uint) : ce paramètre indique le canal de couleur de l'objet bitmap auquel le motif de bruit sera appliqué. Ce nombre peut être une combinaison quelconque des quatre valeurs des canaux de couleur (ARVB). La valeur par défaut est 7.
- `grayScale` (Boolean) : lorsque ce paramètre est activé (`true`), la valeur de `randomSeed` est appliquée aux pixels du bitmap, ce qui donne concrètement un effet de délavé sur toutes les couleurs de l'image. Le canal alpha n'est pas affecté par ce paramètre. La valeur par défaut est `false`.

L'exemple suivant crée une image bitmap et lui applique un motif de bruit bleu :

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmap:BitmapData = new BitmapData(250, 250,false, 0xff000000);
myBitmap.noise(500, 0, 255, BitmapDataChannel.BLUE,false);
var image:Bitmap = new Bitmap(myBitmap);
addChild(image);
```

Si vous souhaitez créer une texture d'aspect plus organique, utilisez la méthode `perlinNoise()` . La méthode `perlinNoise()` produit des textures organiques plus réalistes, qui sont idéales pour des effets de fumée, de nuage, d'eau, de flamme ou même d'explosion.

Comme la méthode `perlinNoise()` fait appel à un algorithme, elle nécessite moins de mémoire vive que les textures à base de bitmaps. Toutefois, elle peut malgré tout consommer beaucoup de ressources processeur, ce qui risque de ralentir le contenu créé par Flash et de provoquer des actualisations d'écran plus lentes que la cadence nominale, en particulier sur les ordinateurs plus anciens. En effet, les algorithmes de cette méthode effectuent des calculs en virgule flottante.

Cette méthode accepte neuf paramètres (les six premiers sont obligatoires) :

- `baseX` (Number) : détermine la valeur x (taille) des motifs créés.
- `baseY` (Number) : détermine la valeur y (taille) des motifs créés.
- `numOctaves:` (uint) : nombre d'octaves ou fonctions de bruit individuelles à combiner pour créer ce bruit. Les nombres d'octaves élevés offrent davantage de détails mais nécessitent également un temps de traitement plus important.
- `randomSeed` (int) : la valeur aléatoire de départ fonctionne exactement comme dans la fonction `noise()`. Pour obtenir un résultat véritablement aléatoire, utilisez la méthode `Math.random()` pour transmettre un nombre aléatoire pour ce paramètre.
- `stitch` (Boolean) : si la valeur est `true`, cette méthode tente de lisser les bords de transition de l'image pour créer des textures sans bords définis, en vue d'une utilisation en mosaïque.

- `fractalNoise` (Boolean) : ce paramètre concerne les bords des dégradés générés par cette méthode. S'il a la valeur `true`, la méthode génère un bruit fractal qui adoucit le pourtour. S'il a la valeur `false`, la méthode génère des turbulences. Les dégradés d'une image créée avec des turbulences présentent des discontinuités visibles qui permettent de mieux restituer certains effets visuels comme les flammes ou les vagues.
- `channelOptions` (uint) : le paramètre `channelOptions` fonctionne exactement comme dans la méthode `noise()`. Il indique le canal de couleur de l'objet bitmap auquel le motif de bruit sera appliqué. Ce nombre peut être une combinaison quelconque des quatre valeurs des canaux de couleur (ARVB). La valeur par défaut est 7.
- `grayScale` (Boolean) : le paramètre `grayScale` fonctionne exactement comme dans la méthode `noise()`. Si ce paramètre est activé (`true`), la valeur de `randomSeed` est appliquée aux pixels du bitmap, ce qui donne concrètement un effet de délavé sur toutes les couleurs de l'image. La valeur par défaut est `false`.
- `offsets:` (Array) : un tableau de points correspondant aux décalages x et y pour chaque octave. En modifiant les valeurs de décalage, vous pouvez faire défiler en continu les calques d'une image. Chaque point du tableau de décalage affecte une fonction de bruit pour une octave spécifique. La valeur par défaut est `null`.

L'exemple suivant crée un objet BitmapData de 150 x 150 pixels qui appelle la méthode `perlinNoise()` pour générer un effet de nuage vert et bleu :

```
import flash.display.Bitmap;
import flash.display.BitmapData;

var myBitmapDataObject:BitmapData = new BitmapData(150, 150, false, 0x00FF0000);

var seed:Number = Math.floor(Math.random() * 100);
var channels:uint = BitmapDataChannel.GREEN | BitmapDataChannel.BLUE
myBitmapDataObject.perlinNoise(100, 80, 6, seed, false, true, channels, false, null);

var myBitmap:Bitmap = new Bitmap(myBitmapDataObject);
addChild(myBitmap);
```

Défilement du contenu d'images bitmap

Supposons que vous avez créé une application cartographique. Chaque fois que l'utilisateur déplace la carte, vous devez actualiser l'affichage (même si le plan n'a été déplacé que de quelques pixels).

Pour obtenir cette fonctionnalité, il est possible de réafficher une nouvelle image contenant le plan actualisé à chaque déplacement. Mais il est également possible de créer une grande image globale et d'utiliser la méthode `scroll()`.

La méthode `scroll()` copie une image bitmap affichée et la colle à un nouvel emplacement, spécifié par les paramètres (`x`, `y`). S'il se trouve qu'une partie de l'image est située hors écran, l'effet obtenu est celui d'un défilement de l'image. Si cette fonction est combinée avec un timer (ou un événement `enterFrame`), l'image semble animée.

L'exemple suivant reprend l'exemple précédent et génère une image bitmap de grande taille (dont les trois-quarts sont restitués hors scène). La méthode `scroll()` est alors appliquée. Grâce à un écouteur pour l'événement `enterFrame`, l'image est décalée d'un pixel en diagonale vers le bas. Cette méthode est appelée pour chaque nouvelle image dans la scène. Les parties de l'image non affichées initialement apparaissent alors peu à peu grâce à ce défilement.

```
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);
}
```

Utilisation du mipmapping

Les *mipmaps* sont des images bitmap qui sont regroupées et associées à une texture dans le but d'améliorer la qualité et les performances d'affichage à l'exécution. Flash Player 9.0.115.0 (et les versions ultérieures) et AIR implémentent cette technique (dite de *mipmapping*), en créant des versions optimisées à diverses échelles de chaque bitmap (en partant de 50 %).

Flash Player et AIR créent des mipmaps pour images bitmap (fichiers JPEG, GIF ou PNG) que vous affichez à l'aide de la classe Loader d'ActionScript 3.0, d'un bitmap dans la bibliothèque de l'outil de programmation Flash ou d'un objet BitmapData. Flash Player crée des mipmaps pour les bitmaps que vous affichez pour la fonction `loadMovie()` d'ActionScript 2.0.

Les mipmaps ne sont pas appliqués aux objets filtrés ni aux clips dont les bitmaps sont en cache. En revanche, ils sont appliqués si un objet d'affichage filtré contient des transformations de bitmap, même si le bitmap se trouve dans un contenu masqué.

Dans Flash Player et AIR, le mipmapping est exécuté automatiquement, mais les quelques conseils suivants vous permettront d'être certain que vos images bénéficient de cette optimisation :

- Pour la lecture vidéo, définissez la propriété `smoothing` sur `true` pour l'objet Video (voir la classe Video).
- Pour les bitmaps, il n'est pas nécessaire de définir la propriété `smoothing` sur `true`, mais l'activation de cette propriété assure une amélioration visible de la qualité.
- Utilisez des tailles de bitmap divisibles par 4 ou 8 pour les images bidimensionnelles (affichage 640 x 128, qui peut être réduit comme suit : 320 x 64 > 160 x 32 > 80 x 16 > 40 x 8 > 20 x 4 > 10 x 2 > 5 x 1) et 2^n pour les textures tridimensionnelles. Les mipmaps sont générés à partir de bitmaps dont la largeur et la hauteur correspondent à 2^n (par ex. 256 x 256, 512 x 512, 1024 x 1024). Le mipmapping s'arrête lorsque Flash Player ou AIR détectent une hauteur ou une largeur impaire.

Exemple : lune en rotation animée

L'exemple de lune en rotation animée illustre les techniques d'utilisation des objets Bitmap et des données d'image bitmap (objets BitmapData). L'exemple crée une animation d'une lune sphérique en rotation et utilise comme données d'image brutes une image plane de la surface de la lune. Les techniques suivantes sont illustrées :

- Chargement d'une image externe et accès aux données d'image brutes correspondantes
- Création d'une animation par copie répétée des pixels de différentes parties d'une image source
- Création d'une image bitmap par définition de la valeur des pixels

Pour obtenir les fichiers d'application de cet exemple, voir www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application de la lune en rotation animée résident dans le dossier Samples/SpinningMoon. L'application se compose des fichiers suivants :

Fichier	Description
SpinningMoon.mxml ou SpinningMoon.fla	Le fichier d'application principal dans Flex (MXML) ou Flash (FLA).
com/example/programmingas3/moon/MoonSphere.as	Classe exécutant le chargement, l'affichage et l'animation de la lune.
moonMap.png	Fichier image contenant une photographie de la surface de la lune, chargé et utilisé pour créer la lune en rotation animée.

Chargement d'une image externe comme données bitmap

La première tâche à exécuter dans cet exemple consiste à charger une image externe, une photographie de la surface de la lune. Le chargement est géré par deux méthodes de la classe MoonSphere : le constructeur `MoonSphere()`, qui lance le processus de chargement, et la méthode `imageLoadComplete()`, qui est appelée au terme du chargement de l'image externe.

Le chargement d'une image externe est similaire à celui d'un fichier SWF externe : il est réalisé par une occurrence de la classe flash.display.Loader. Le code ci-après de la méthode `MoonSphere()` commence à charger l'image :

```
var imageLoader:Loader = new Loader();
imageLoader.contentLoaderInfo.addEventListener(Event.COMPLETE, imageLoadComplete);
imageLoader.load(new URLRequest("moonMap.png"));
```

La première ligne déclare l'occurrence de Loader appelée `imageLoader`. La troisième ligne commence le processus de chargement à proprement parler en appelant la méthode `load()` de l'objet Loader. Cette méthode transmet une occurrence URLRequest représentant l'URL de l'image à charger. La deuxième ligne définit l'écouteur d'événement qui se déclenchera à l'issue du chargement de l'image. Observez que la méthode `addEventListener()` n'est pas appelée sur l'occurrence de Loader elle-même, mais sur la propriété `contentLoaderInfo` de l'objet Loader. L'occurrence de Loader n'envoie pas d'événements en rapport avec le contenu chargé. En revanche, sa propriété `contentLoaderInfo` contient une référence à l'objet LoaderInfo qui est associé au contenu chargé dans l'objet Loader (en l'occurrence, l'image externe). L'objet LoaderInfo génère des événements en rapport avec le déroulement et la fin du chargement du contenu externe, notamment l'événement `complete` (`Event.COMPLETE`) qui déclenche un appel à la méthode `imageLoadComplete()` au terme du chargement de l'image.

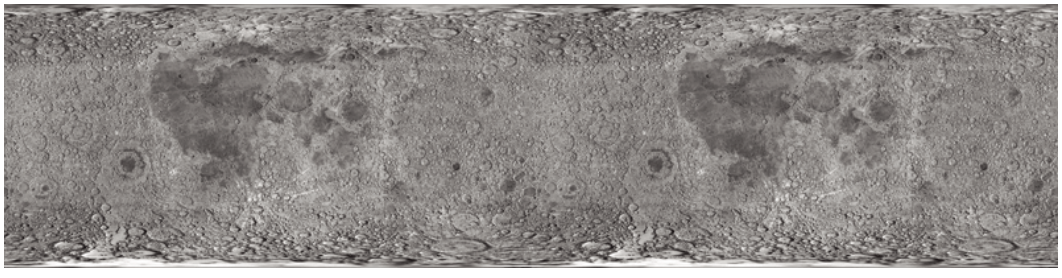
S'il est essentiel de lancer le chargement de l'image externe, il est tout aussi important de savoir comment procéder au terme de cette opération. Comme l'illustre le code ci-dessus, la fonction `imageLoadComplete()` est appelée une fois l'image chargée. Cette fonction exécute diverses opérations sur les données chargées, ainsi que l'indiquent les sections suivantes. Cependant, pour utiliser les données d'image, elle doit pouvoir y accéder. Une image externe chargée par le biais d'un objet Loader devient une image Bitmap jointe en tant qu'objet d'affichage enfant de l'objet Loader. Dans ce cas, la méthode écouteur d'événement a accès à l'occurrence de Loader dans l'objet événement transmis en tant que paramètre à la méthode. Les premières lignes de la méthode `imageLoadComplete()` sont les suivantes :

```
private function imageLoadComplete(event:Event):void
{
    textureMap = event.target.content.bitmapData;
    ...
}
```

Notez que le paramètre de l'objet événement s'appelle `event` et que c'est une occurrence de la classe Event. Chaque occurrence de la classe Event possède une propriété `target`, qui fait référence à l'objet déclenchant l'événement (en l'occurrence, l'occurrence de LoaderInfo sur laquelle la méthode `addEventListener()` a été appelée, comme indiqué plus haut). De même, l'objet LoaderInfo possède une propriété `content` qui, à l'issue du chargement, contient une occurrence de Bitmap comportant l'image bitmap chargée. Pour afficher l'image directement à l'écran, vous pouvez joindre cette occurrence de Bitmap (`event.target.content`) à un conteneur d'objet d'affichage. (Vous pouvez aussi joindre l'objet Loader à un conteneur d'objet d'affichage.) Toutefois, dans cet exemple, le contenu chargé n'est pas affiché à l'écran, il est utilisé en tant que données d'image brutes. Par conséquent, la première ligne de la méthode `imageLoadComplete()` lit la propriété `bitmapData` de l'occurrence de Bitmap chargée (`event.target.content.bitmapData`) et la stocke dans la variable d'occurrence de `textureMap`, qui, comme le décrit la section suivante, est utilisée en tant que source des données d'image pour créer l'animation de la lune en rotation.

Création d'une animation par copie de pixels

Une animation, dans sa définition la plus simple, est l'illusion d'un mouvement ou d'un changement, créée par la modification graduelle d'une image. Cet exemple a pour but de créer l'illusion d'une lune sphérique tournant sur son axe vertical. Cependant, pour les besoins de l'animation, vous pouvez ne pas tenir compte de l'aspect de distorsion sphérique de l'exemple. Examinez l'image chargée et utilisée comme source des données d'image de la lune :



Comme vous pouvez le constater, l'image ne représente pas une ou plusieurs sphères ; c'est une photographie rectangulaire de la surface de la lune. La photographie ayant été prise à l'emplacement exact de l'équateur de la lune, les parties supérieures et inférieures de l'image sont donc étirées et déformées. Pour supprimer la distorsion de l'image et lui redonner son aspect sphérique, nous utiliserons un filtre Mappage de déplacement (voir plus bas). Toutefois, comme l'image source est un rectangle, pour créer l'illusion d'une sphère en rotation, il suffit que le code fasse glisser la photographie de la surface de la lune horizontalement, comme l'indiquent les paragraphes suivants.

Observez que l'image contient en fait deux copies juxtaposées de la photographie de la surface de la lune. Cette image représente l'image source dans laquelle des données ont été copiées plusieurs fois pour créer un effet de mouvement. La juxtaposition de deux copies de l'image facilite la création d'un effet de défilement continu. Examinons en détail le processus d'animation afin de mieux le comprendre.

Le processus s'applique à deux objets ActionScript distincts. Le premier de ces objets est l'image source chargée qui, dans le code, est représentée par l'occurrence de BitmapData `textureMap` Comme nous l'avons vu, les données d'image sont insérées dans `textureMap` dès le chargement de l'image externe à l'aide de ce code :

```
textureMap = event.target.content.bitmapData;
```

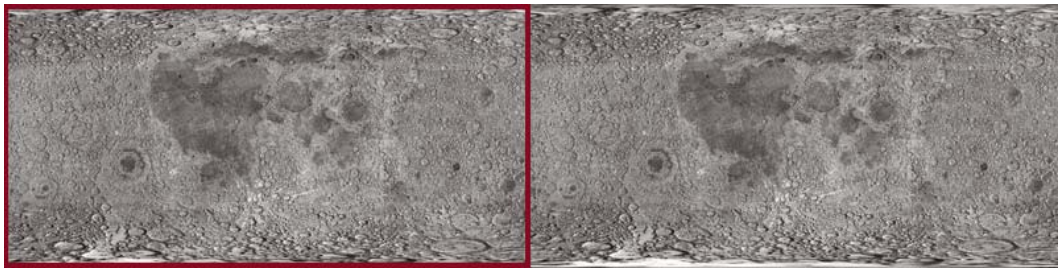
`textureMap` contient l'image illustrée au préalable. En outre, pour créer la rotation animée, l'exemple utilise l'occurrence de Bitmap `sphere`, qui représente l'objet d'affichage qui affiche l'image de la lune à l'écran. A l'instar de `textureMap`, l'objet `sphere` contient les données d'image initiales de la méthode `imageLoadComplete()`, ainsi que le stipule le code suivant :

```
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));
```

Comme vous pouvez le constater, `sphere` est instancié. La hauteur et la largeur de sa propriété `bitmapData` (les données d'image brutes qui sont affichées par `sphere`) sont identiques à celles de `textureMap`. Autrement dit, le contenu de `sphere` a la même taille qu'une seule photographie de la lune (puisque l'image `textureMap` contient deux photographies juxtaposées). Des données d'image sont ensuite insérées dans la propriété `bitmapData` à l'aide de sa méthode `copyPixels()`. Les paramètres de l'appel de la méthode `copyPixels()` donnent plusieurs indications :

- Le premier paramètre indique que les données d'image copiées proviennent de `textureMap`.
- Le deuxième paramètre, une nouvelle occurrence de Rectangle, détermine quelle partie de `textureMap` est copiée. En l'occurrence, le cliché est un rectangle dont l'origine coïncide avec le coin supérieur gauche de `textureMap` (ce qu'indiquent les deux premiers paramètres de `Rectangle() : 0, 0`) et dont la largeur et la hauteur correspondent aux propriétés `width` et `height` de `sphere`.
- Le troisième paramètre, une nouvelle occurrence de Point avec des valeurs de x et y égales à `0`, définit la destination des données de pixel ; en l'occurrence, le coin supérieur gauche (0, 0) de `sphere.bitmapData`.

Représenté visuellement, le code copie les pixels de `textureMap` mis en évidence ci-dessous et les colle sur `sphere`. Autrement dit le contenu BitmapData de `sphere` correspond à la partie de `textureMap` mise en évidence :



Pour rappel, il s'agit seulement de l'état initial de `sphere`, le contenu de la première image copiée sur `sphere`.

Une fois l'image source chargée et `sphere` créé, il ne reste plus à la méthode `imageLoadComplete()` qu'à définir l'animation. L'animation est pilotée par une occurrence de Timer, `rotationTimer`, créée et lancée par le code suivant :

```
var rotationTimer:Timer = new Timer(15);
rotationTimer.addEventListener(TimerEvent.TIMER, rotateMoon);
rotationTimer.start();
```

Le code commence par créer l'occurrence de Timer `rotationTimer`. Le paramètre passé au constructeur `Timer()` indique que `rotationTimer` doit déclencher son événement `timer` toutes les 15 millisecondes. La méthode `addEventListener()` qui est appelée ensuite stipule que le déclenchement de l'événement `timer` (`TimerEvent.TIMER`) entraîne l'appel de la méthode `rotateMoon()`. Enfin, l'appel de la méthode `start()` du timer entraîne le démarrage de celui-ci.

De par la définition de `rotationTimer`, Flash Player appelle la méthode `rotateMoon()` dans la classe MoonSphere environ toutes les 15 millisecondes, ce qui se traduit par l'animation de la lune. Le code source de la méthode rotateMoon() est le suivant :

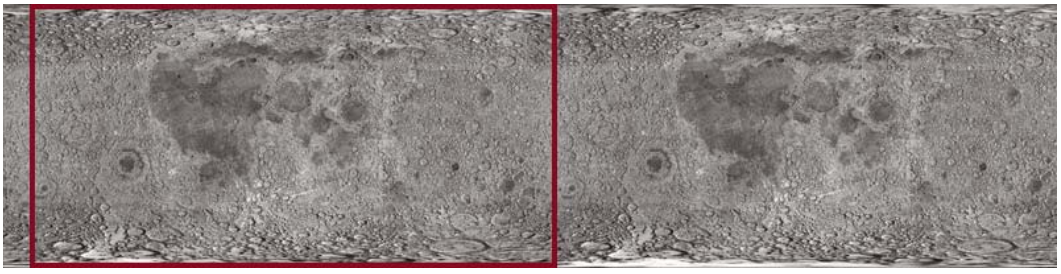
```
private function rotateMoon(event:TimerEvent):void
{
    sourceX += 1;
    if (sourceX > textureMap.width / 2)
    {
        sourceX = 0;
    }

    sphere.bitmapData.copyPixels(textureMap,
                                 new Rectangle(sourceX, 0, sphere.width, sphere.height),
                                 new Point(0, 0));

    event.updateAfterEvent();
}
```

Ce code effectue trois opérations :

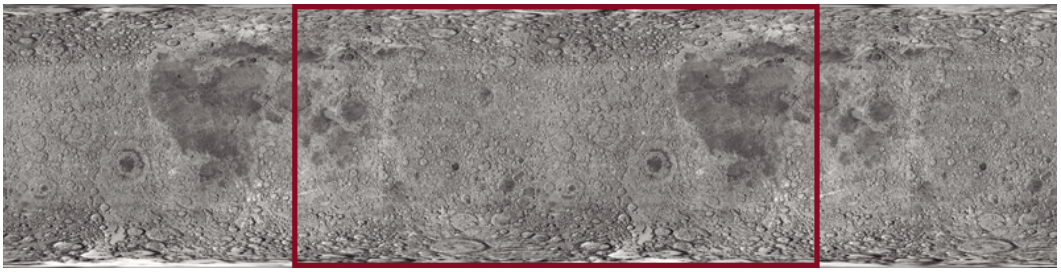
1 La valeur de la variable `sourceX` (initialement fixée à 0) est incrémentée d'une unité.

```
sourceX += 1;
```

`sourceX` permet de déterminer d'où proviennent, dans `textureMap`, les pixels copiés sur `sphere`. Ce code déplace donc le rectangle d'un pixel vers la droite sur `textureMap`. Comme le montre l'illustration suivante, après plusieurs cycles d'animation, le rectangle source s'est déplacé de plusieurs pixels vers la droite :

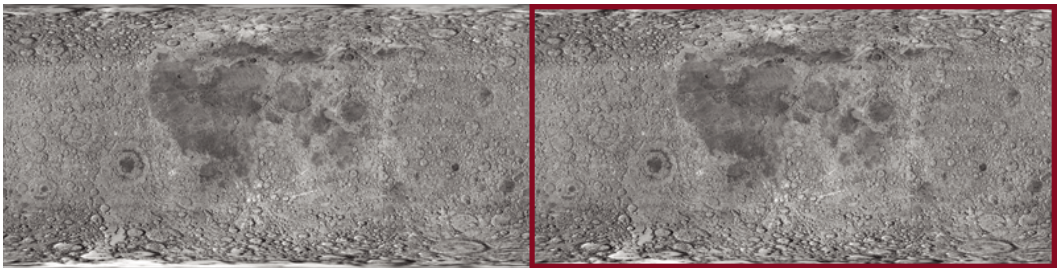


Après plusieurs autres cycles, le rectangle se trouve encore plus à droite.



C'est sur ce déplacement progressif constant de l'emplacement d'origine des pixels copiés que repose l'animation. Par un déplacement lent mais continu de l'emplacement source vers la droite, l'image affichée à l'écran dans `sphere` semble continuellement glisser vers la gauche. C'est pourquoi l'image source (`textureMap`) doit contenir deux copies de la photographie de la surface de la lune. Comme le rectangle se déplace continuellement vers la droite, il chevauche généralement les deux photographies et non pas seulement l'une d'elles.

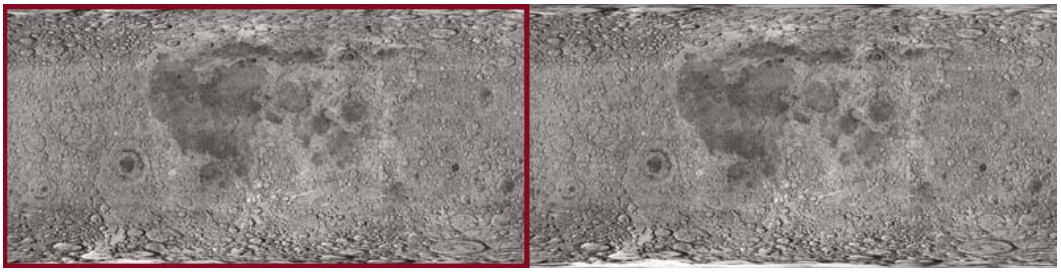
2 Ce lent déplacement vers la droite donne cependant lieu à un problème. Le rectangle finira par atteindre le bord droit de `textureMap` et ne trouvera plus de pixels à copier sur `sphere`:



Les lignes suivantes du code permettent de résoudre ce problème :

```
if (sourceX >= textureMap.width / 2)
{
    sourceX = 0;
}
```

Le code vérifie si `sourceX` (le bord gauche du rectangle) a atteint le milieu de `textureMap`. Si tel est le cas, il remet la variable `sourceX` à 0 ; autrement dit, il la ramène au bord gauche de `textureMap`, et le cycle recommence :



3 Une fois la valeur de `sourceX` appropriée calculée, la dernière étape du processus d'animation consiste à copier les pixels du nouveau rectangle source sur `sphere`. Pour ce faire, nous reprenons le code qui a initialement rempli `sphere` (voir plus haut), à la différence près que, dans l'appel du constructeur `new Rectangle()`, le bord gauche du rectangle est placé à `sourceX` :

```
sphere.bitmapData.copyPixels(textureMap,
                             new Rectangle(sourceX, 0, sphere.width, sphere.height),
                             new Point(0, 0));
```

Pour rappel, ce code est appelé toutes les 15 millisecondes. Comment l'emplacement du rectangle source change constamment et que les pixels sont copiés sur `sphere`, à l'écran, la photographie de la lune représentée par `sphere` semble glisser continuellement. En d'autres termes, la lune semble tourner sur elle-même continuellement.

Définition de l'aspect sphérique

La lune est bien entendu sphérique , ce n'est pas un rectangle. La photographie rectangulaire de la surface lunaire, qui fait l'objet d'une animation constante, doit donc être convertie en sphère. Cette opération comprend deux étapes : un masque cache tout le contenu excepté une partie circulaire de la photographie et un filtre Mappage de déplacement déforme l'apparence de la photographie, lui donnant un aspect tridimensionnel.

Dans un premier temps, un masque circulaire cache entièrement le contenu de l'objet MoonSphere excepté la sphère créée par le filtre. Le code suivant crée le masque, une occurrence de Shape, et l'applique à l'occurrence de MoonSphere :

```
moonMask = new Shape();
moonMask.graphics.beginFill(0);
moonMask.graphics.drawCircle(0, 0, radius);
this.addChild(moonMask);
this.mask = moonMask;
```

Comme MoonSphere est un objet d'affichage (fondé sur la classe Sprite), il est possible d'appliquer directement le masque à l'occurrence de MoonSphere à l'aide de sa propriété `mask` héritée.

Il ne suffit pas d'occulter des parties de la photographie à l'aide d'un masque circulaire pour créer un effet réaliste de sphère en rotation. En raison de la façon dont la photographie de la surface lunaire a été prise, ses dimensions ne sont pas proportionnelles . les parties supérieures et inférieures de l'image sont déformées et étirées par rapport aux zones équatoriales. Pour déformer l'apparence de la photographie et lui donner un aspect tridimensionnel, nous allons utiliser un filtre Mappage de déplacement.

Ce type de filtre permet de déformer une image. En l'occurrence, nous allons déformer la photographie de la lune pour lui donner un aspect plus réaliste, en compressant horizontalement les parties supérieures et inférieures de l'image sans toucher à son milieu. En supposant que le filtre intervienne sur une partie carrée de la photographie, la compression du haut et du bas mais pas du milieu aura pour effet de convertir le carré en cercle. L'animation de cette image déformée a un effet secondaire : la distance en pixels parcourue par le milieu de l'image semble supérieure à celle couverte par les parties supérieure et inférieure, d'où l'impression que le cercle est en fait un objet tridimensionnel (une sphère).

Le code suivant permet de créer un filtre Mappage de déplacement appelé `displaceFilter` :

```
var displaceFilter:DisplacementMapFilter;
displaceFilter = new DisplacementMapFilter(fisheyeLens,
                                new Point(radius, 0),
                                BitmapDataChannel.RED,
                                BitmapDataChannel.GREEN,
                                radius, 0);
```

Le premier paramètre, `fisheyeLens`, est l'image de mappage ; en l'occurrence, un objet BitmapData créé par programmation. La création de cette image est décrite ci-dessous, à la section « Création d'une image bitmap par définition de la valeur des pixels » à la page 512. Les autres paramètres décrivent l'emplacement d'application du filtre au sein de l'image filtrée, les canaux colorimétriques utilisés pour régir l'effet de déplacement et leur impact sur celui-ci. Une fois le filtre Mappage de déplacement créé, il est appliqué à `sphere`, toujours dans la méthode `imageLoadComplete()` :

```
sphere.filters = [displaceFilter];
```

L'image finale, une fois le masque et le filtre appliqués, se présente comme suit :

A chaque cycle du processus d'animation de la lune en rotation, le contenu BitmapData de sphere est remplacé par un nouveau cliché des données d'image source. Il est cependant inutile de réappliquer le filtre à chaque fois car il est appliqué à l'occurrence de Bitmap (l'objet d'affichage) plutôt qu'aux données bitmap (données de pixel brutes). Pour rappel, l'occurrence de Bitmap ne correspond pas aux données bitmap. C'est un objet d'affichage qui affiche ces données à l'écran. Une occurrence de Bitmap peut être assimilée à un projecteur de diapositives, tandis qu'un objet BitmapData serait une diapositive présentée par le biais du projecteur. Il est possible d'appliquer un filtre directement à un objet BitmapData, ce qui reviendrait à dessiner sur une diapositive pour modifier l'image. Vous pouvez aussi

appliquer un filtre à tout objet d'affichage, y compris une occurrence de Bitmap, ce qui équivaudrait à placer un filtre devant l'objectif du projecteur pour déformer l'image à l'écran sans modifier la diapositive d'origine. Comme les données bitmap brutes sont accessibles par le biais de la propriété bitmapData d'une occurrence de Bitmap, rien n'empêche de leur appliquer directement le filtre. Dans ce cas, cependant, il est préférable d'appliquer directement le filtre à l'objet d'affichage Bitmap plutôt qu'aux données bitmap.

Pour plus d'informations sur l'utilisation du filtre Mappage de déplacement en ActionScript, consultez le chapitre « Filtrage des objets d'affichage » à la page 363.

Création d'une image bitmap par définition de la valeur des pixels

Le fait qu'un filtre Mappage de déplacement implique en réalité deux images est un facteur important. L'image source est modifiée par le filtre. Dans cet exemple, il s'agit de l'occurrence de Bitmap `sphere`. L'autre image utilisée par le filtre est appelée l'image de mappage. Elle n'apparaît pas à l'écran. En revanche, la couleur de ses pixels est utilisée en entrée par la fonction de déplacement : la couleur d'un pixel se trouvant à des coordonnées x, y spécifiques détermine le déplacement (changement physique de position) à appliquer au pixel à ces coordonnées x, y dans l'image source.

Dans cet exemple, pour utiliser le filtre Mappage de déplacement en vue de créer un effet sphérique, il est donc nécessaire d'utiliser l'image de mappage appropriée, c'est-à-dire une image au fond gris comportant un cercle rempli d'un dégradé d'une seule couleur (rouge) qui passe, horizontalement, du foncé au clair, comme illustré ci-dessous :

Comme une image de mappage et un filtre uniques sont utilisés dans cet exemple, l'image de mappage est créée une seule fois, dans la méthode `imageLoadComplete()` (autrement dit, à l'issue du chargement de l'image externe). L'image de mappage, `fisheyeLens`, est créée par appel de la méthode `createFisheyeMap()` de la classe MoonSphere :

```
var fisheyeLens:BitmapData = createFisheyeMap(radius);
```

Au sein de la méthode `createFisheyeMap()`, l'image de mappage est dessinée pixel par pixel à l'aide de la méthode `setPixel()` de la classe BitmapData. Vous trouverez le code complet de la méthode `createFisheyeMap()` ci-dessous, suivi d'une présentation détaillée de son fonctionnement :

```
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;
}
```

En premier lieu, la méthode reçoit un paramètre, `radius`, qui indique le rayon de l'image circulaire à créer. Le code crée ensuite l'objet BitmapData sur lequel sera tracé le cercle. Cet objet, appelé `result`, est renvoyé comme valeur résultante de la méthode. Comme illustré par l'extrait de code ci-dessous, la largeur et la hauteur de l'occurrence de BitmapData `result` créée sont égales au diamètre du cercle. En outre, cette occurrence n'a pas de transparence (le troisième paramètre correspond à `false`) et elle est pré-remplie par la couleur `0x808080` (gris moyen) :

```
var result:BitmapData = new BitmapData(diameter,
                                       diameter,
                                       false,
                                       0x808080);
```

Le code utilise ensuite deux boucles pour itérer par-dessus chaque pixel de l'image. La boucle extérieure parcourt de droite à gauche chaque colonne de l'image (la variable `i` représentant la position horizontale du pixel manipulé), alors que la boucle intérieure intervient sur chaque pixel de la colonne actuelle, de bas en haut (la variable `j` représentant la position verticale du pixel actuel). Le code des boucles (le contenu de la boucle intérieure étant omis) est illustré ci-dessous :

```
for (var i:int = 0; i < diameter; i++)
{
    for (var j:int = 0; j < diameter; j++)
    {
        ...
    }
}
```

Au fur et à mesure de la manipulation des pixels par les boucles, une valeur est calculée à chacun d'eux (la valeur colorimétrique de ce pixel dans l'image de mappage). Ce processus comporte quatre étapes :

1. Le code calcule la distance séparant le pixel actuel du centre du cercle, le long de l'axe x (`i - radius`). Cette valeur est divisée par le rayon pour obtenir un pourcentage de celui-ci plutôt qu'une distance absolue (`(i - radius) / radius`). Ce pourcentage est stocké dans une variable appelée `pctX`. La valeur équivalente sur l'axe y est calculée et stockée dans la variable `pctY`, comme illustré dans le code ci-dessous :

   ```
   var pctX:Number = (i - radius) / radius;
   var pctY:Number = (j - radius) / radius;
   ```

2. Une formule trigonométrique standard (le théorème de Pythagore) est utilisée pour calculer la distance linéaire entre le centre du cercle et le point actuel, à partir de `pctX` et `pctY`. Cette valeur est stockée dans une variable, `pctDistance`, comme illustré ci-dessous :

   ```
   var pctDistance:Number = Math.sqrt(pctX * pctX + pctY * pctY);
   ```

3. Le code vérifie ensuite si la distance en pourcentage est inférieure à 1 (ou 100 % du rayon ; autrement dit, si le pixel concerné se trouve sur le rayon du cercle). Si le pixel figure dans le cercle, une valeur colorimétrique calculée lui est affectée (voir la description à l'étape 4). Dans le cas contraire, ce pixel ne fait l'objet d'aucune manipulation et conserve la couleur par défaut, c'est-à-dire le gris moyen.

   ```
   if (pctDistance < 1)
   {
       ...
   }
   ```

4. Une valeur colorimétrique est calculée pour tout pixel se trouvant dans le cercle. La couleur finale est une nuance de rouge qui va du noir (0 % de rouge) sur le bord gauche du cercle au rouge vif (100 % de rouge) sur le bord droit du cercle. La valeur colorimétrique comprend initialement trois parts de couleur (rouge, vert et bleu), comme illustré ci-dessous) :

   ```
   red = 128 * (1 + 0.75 * pctX * pctX * pctX / (1 - pctY * pctY));
   green = 0;
   blue = 0;
   ```

Observez que seule la part rouge de la couleur (variable `red`) possède une valeur. Les valeurs vert et bleu (variables `green` et `blue`) sont illustrées ici par souci de clarté, mais il est possible de les omettre. Cette méthode ayant pour but de créer un cercle contenant un dégradé de rouge, les valeurs vertes et bleues sont superflues.

Une fois les trois valeurs colorimétriques individuelles déterminées, elles sont conjuguées dans une valeur entière unique à l'aide d'un algorithme de décalage de bits, comme illustré ci-dessous :

```
rgb = (red << 16 | green << 8 | blue);
```

En dernier lieu, la valeur colorimétrique calculée est affectée au pixel actuel à l'aide de la méthode `setPixel()` de l'objet BitmapData `result`, comme illustré ci-dessous :

```
result.setPixel(i, j, rgb);
```

Chapitre 23 : Travail en trois dimensions (3D)

Principes de base de la 3D

Introduction à la 3D dans ActionScript

La principale différence entre un objet en deux dimensions (2D) et un objet en trois dimensions (3D) projeté sur un écran en deux dimensions consiste en l'ajout d'une troisième dimension à l'objet. La troisième dimension permet à l'objet de se rapprocher et de s'éloigner du point de vue de l'utilisateur.

Lorsque vous définissez explicitement la propriété `z` d'un objet d'affichage sur une valeur numérique, l'objet crée automatiquement une matrice de transformation 3D. Vous pouvez intervenir sur cette matrice pour modifier les paramètres de transformation 3D de l'objet.

En outre, la rotation 3D diffère de la rotation 2D. En 2D, l'axe de la rotation est systématiquement perpendiculaire au plan x/y, autrement dit, elle se trouve sur l'axe z. En 3D, l'axe de rotation peut se trouver autour de n'importe lequel des axes, x, y ou z. La définition des propriétés de rotation et de mise à l'échelle d'un objet d'affichage lui permet de se déplacer dans l'espace 3D.

Tâches 3D courantes

Les tâches 3D courantes ci-dessous sont décrites dans ce chapitre :

- Création d'un objet 3D
- Déplacement d'un objet dans l'espace 3D
- Rotation d'un objet dans l'espace 3D
- Représentation de la profondeur à l'aide d'une projection de perspective
- Réorganisation de la liste d'affichage pour qu'elle corresponde à des axes z relatifs afin que les objets s'affichent correctement l'un devant l'autre
- Transformation d'objets 3D à l'aide de matrices 3D
- Manipulation d'objets dans l'espace 3D à l'aide de vecteurs
- Création d'une perspective à l'aide de la méthode `Graphics.drawTriangles()`
- Ajout de textures bitmap à un objet 3D par le biais du mappage des coordonnées UV
- Définition du paramètre culling de la méthode `Graphics.drawTriangles()` pour accélérer le rendu et masquer les parties d'un objet 3D qui ne sont pas visibles à partir du point de vue actuel

Terminologie et concepts importants

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Perspective : dans un plan 2D, représentation de lignes parallèles convergeant vers un point de fuite pour donner une illusion de profondeur et de distance
- Projection : génération d'une image 2D d'un objet 3D ou plus. La projection 3D mappe des points 3D sur un plan 2D

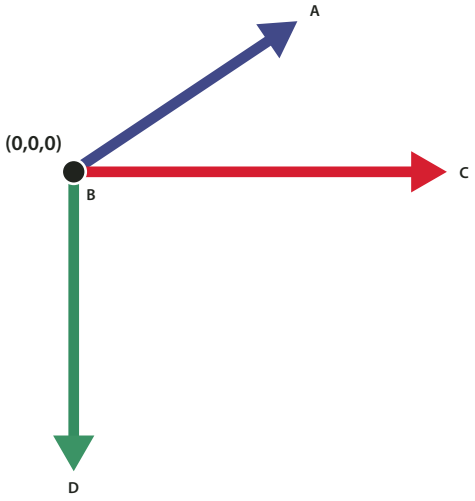
- Rotation : modification de l'orientation (et souvent de la position) d'un objet en faisant décrire un mouvement circulaire à chacun de ses points
- Transformation : modification de points 3D ou d'ensemble de points par translation, rotation, mise à l'échelle, inclinaison ou une combinaison de ces actions
- Translation : changement de la position d'un objet en déplaçant chacun de ses points sur une distance et dans une direction identiques
- Point de fuite : point auquel des lignes parallèles qui s'éloignent semblent se rencontrer lorsqu'elles sont représentées dans une perspective linéaire
- Vecteur : un vecteur 3D représente un point ou un emplacement dans l'espace en trois dimensions à l'aide de coordonnées cartésiennes (x,y,z).
- Sommet : point d'angle
- Maillage texturé : tout point définissant un objet dans l'espace 3D
- Mappage des coordonnées UV : mode d'application d'une texture ou d'une bitmap à une surface 3D. Le mappage des coordonnées UV affecte des valeurs à des coordonnées sur une image en tant que pourcentages de l'axe horizontal (U) et de l'axe vertical (V).
- Valeur T : facteur de mise à l'échelle permettant de déterminer la taille d'un objet 3D lorsque celui-ci se rapproche ou s'éloigne du point de vue actuel
- Culling : rendu, ou non, des surfaces avec un enroulement spécifique. L'utilisation du culling (élimination) permet de masquer des surfaces qui ne sont pas visibles à partir du point de vue actuel.

Description des fonctions 3D de Flash Player et du moteur d'exécution AIR

Dans les versions de Flash Player antérieures à Flash Player 10 et dans les versions d'Adobe AIR antérieures à Adobe AIR 1.5, les objets d'affichage possèdent deux propriétés, `x` et `y`, permettant de positionner ces derniers sur un plan 2D. A partir de Flash Player 10 et Adobe AIR 1.5, tout objet d'affichage ActionScript est doté d'une propriété `z` permettant de le positionner le long de l'axe z, qui est généralement utilisé pour indiquer la profondeur ou la distance.

Flash Player 10 et Adobe AIR 1.5 prennent désormais en charge les effets 3D. Les objets 3D restent cependant essentiellement plats. Tout objet d'affichage, tel qu'un objet MovieClip ou Sprite, effectue en fait son propre rendu en deux dimensions, sur un plan unique. Les fonctions 3D vous permettent de placer, déplacer, faire pivoter et transformer de diverses façons ces objets planaires dans la totalité des trois dimensions. Elles vous permettent également de gérer les points 3D et de les convertir en coordonnées 2D (x et y), afin que vous puissiez projeter les objets 3D sur un affichage 2D. A l'aide de ces fonctions, vous pouvez simuler de nombreux types d'effets 3D.

Le système de coordonnées 3D utilisé par ActionScript est différent. Lorsque vous utilisez des coordonnées 2D dans ActionScript, la valeur de x augmente au fur et à mesure du déplacement vers la droite le long de l'axe x, et la valeur de y augmente au fur et à mesure du déplacement vers le bas le long de l'axe y. Le système de coordonnées 3D conserve ces conventions et ajoute un axe z dont la valeur augmente au fur et à mesure que vous vous éloignez du point de vue.



Directions positives des axes x, y et z dans le système de coordonnées 3D
A. *+ axe Z* ***B.*** *Origine* ***C.*** *+ axe X* ***D.*** *+ axe Y*

Remarque : *n'oubliez pas que Flash Player et AIR représentent toujours la 3D en calques. Par conséquent, si l'objet A se trouve devant l'objet B dans la liste d'affichage, Flash Player ou AIR rend toujours A devant B, quelles que soient les valeurs sur l'axe z des deux objets. Pour résoudre le conflit entre l'ordre de la liste d'affichage et celui de l'axe z, utilisez la méthode* `transform.getRelativeMatrix3D()` *afin d'enregistrer, puis de réorganiser les calques des objets d'affichage 3D. Pour plus d'informations, consultez la section « Réorganisation de l'affichage à l'aide d'objets Matrix3D » à la page 527.*

Les classes ActionScript suivantes prennent en charge les nouvelles fonctions 3D :

1. La classe flash.display.DisplayObject contient la propriété `z` et de nouvelles propriétés de rotation et de mise à l'échelle permettant de manipuler les objets d'affichage dans l'espace 3D. La méthode `DisplayObject.local3DToGlobal()` simplifie la projection de géométrie 3D sur un plan 2D.
2. Vous pouvez utiliser la classe flash.geom.Vector3D en tant que structure de données pour la gestion des points 3D. Elle prend aussi en charge la mathématique vectorielle.
3. La classe flash.geom.Matrix3D prend en charge les transformations complexes de géométrie 3D, telles que la rotation, la mise à l'échelle et la translation.
4. La classe flash.geom.PerspectiveProjection contrôle les paramètres de mappage de géométrie 3D sur un affichage 2D.

ActionScript propose deux approches différentes pour simuler des images 3D :

1. Agencement et animation d'objets planaires dans l'espace 3D. Cette approche implique l'animation d'objets d'affichage à l'aide de leurs propriétés x, y et z, ou la définition des propriétés de rotation et de mise à l'échelle par le biais de la classe DisplayObject. Il est possible de générer des mouvements plus complexes à l'aide de l'objet DisplayObject.transform.matrix3D. L'objet DisplayObject.transform.perspectiveProjection personnalise le tracé des objets d'affichage dans la perspective 3D. Adoptez cette approche pour animer des objets 3D principalement composés de plans. Elle convient aux galeries d'image 3D ou aux objets d'animation 2D agencés dans l'espace 3D, par exemple.

2 Génération de triangles 2D à partir d'une géométrie 3D et rendu de ces triangles avec des textures. Pour ce faire, vous devez d'abord définir et gérer des données relatives aux objets 3D, puis les convertir en triangles 2D à des fins de rendu. Il est possible de mapper des textures bitmap sur ces triangles, qui sont ensuite tracés sur un objet graphique à l'aide de la méthode `Graphics.drawTriangles()`. Cette approche est appropriée pour le chargement des données d'un modèle 3D à partir d'un fichier et le rendu du modèle à l'écran, ou pour la génération et le tracé d'un terrain 3D en tant que maillages triangulaires, par exemple.

Création et déplacement d'objets 3D

Pour convertir un objet d'affichage 2D en objet d'affichage 3D, vous devez explicitement définir sa propriété `z` sur une valeur numérique. Lorsque vous affectez une valeur à la propriété `z`, un objet Transform est créé pour l'objet d'affichage. La définition des propriétés `DisplayObject.rotationX` ou `DisplayObject.rotationY` crée également un objet Transform. Celui-ci contient une propriété `Matrix3D` qui régit la représentation de l'objet d'affichage dans l'espace 3D.

Le code suivant définit les coordonnées d'un objet d'affichage appelé « leaf » (feuille) :

```
leaf.x = 100; leaf.y = 50; leaf.z = -30;
```

Vous pouvez visualiser ces valeurs, ainsi que les propriétés qui en dérivent, dans la propriété `matrix3D` de l'objet Transform de la feuille :

```
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);
```

Pour plus d'informations sur les propriétés de l'objet Transform, consultez la classe Transform. Pour plus d'informations sur les propriétés de l'objet Matrix3D, consultez la classe Matrix3D.

Déplacement d'un objet dans l'espace 3D

Vous pouvez déplacer un objet dans l'espace 3D en modifiant la valeur de ses propriétés `x, y ou z`. Lorsque vous modifiez la valeur de sa propriété `z`, l'objet semble se rapprocher ou s'éloigner de l'observateur.

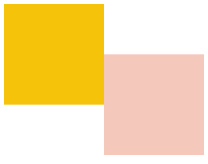
Le code suivant modifie la valeur de la propriété `z` de deux ellipses en réponse à un événement pour leur imprimer un mouvement de va-et-vient le long de leur axe z. `ellipse2` se déplace plus rapidement que `ellipse1` : sa propriété `z` est incrémentée d'un multiple de 20 sur chaque événement Frame, alors que la propriété `z` d'`ellipse1` est incrémentée d'un multiple de 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;
    }
}
```

Rotation d'un objet dans l'espace 3D

Vous pouvez faire pivoter un objet de trois façons différentes, selon la définition de sa propriété de rotation : `rotationX`, `rotationY` et `rotationZ`.

La figure ci-dessous illustre deux carrés qui ne sont pas soumis à une rotation :

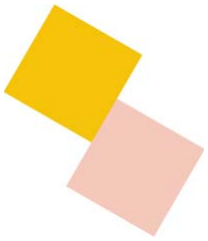


Dans la figure suivante, la propriété `rotationY` du conteneur des carrés a été incrémentée pour les faire pivoter sur l'axe y. La rotation du conteneur, ou objet d'affichage parent, des deux carrés fait pivoter ceux-ci :

```
container.rotationY += 10;
```

Dans la figure suivante, la propriété `rotationX` du conteneur des carrés est modifiée : Elle fait pivoter ceux-ci sur l'axe x.

Dans la figure suivante, la propriété `rotationZ` du conteneur des carrés a été incrémentée et ceux-ci pivotent sur l'axe z.



Un objet d'affichage peut se déplacer et pivoter simultanément dans l'espace 3D.

Projection d'objets 3D sur un affichage 2D

La classe PerspectiveProjection du package `flash.geom` facilite l'application d'une perspective rudimentaire lors du déplacement d'objets d'affichage dans l'espace 3D.

Si vous ne créez pas explicitement une projection de perspective pour votre espace 3D, le moteur 3D utilise un objet PerspectiveProjection par défaut, qui existe sur la racine et est propagé à tous ses enfants.

Les trois propriétés qui définissent le mode d'affichage de l'espace 3D par un objet PerspectiveProjection sont les suivantes :

- `fieldOfView`
- `projectionCenter`
- `focalLength`

La modification de la valeur de `fieldOfView` entraîne automatiquement la modification de la valeur de `focalLength`, et inversement, car ces propriétés sont interdépendantes.

La formule permettant de calculer `focalLength` en fonction de la valeur de `fieldOfView` est la suivante :

```
focalLength = stageWidth/2 * (cos(fieldOfView/2) / sin(fieldOfView/2)
```

En règle générale, vous modifiez explicitement la propriété `fieldOfView`.

Champ de vision

En manipulant la propriété `fieldOfView` de la classe PerspectiveProjection, vous pouvez faire en sorte qu'un objet d'affichage 3D semble s'agrandir ou diminuer selon qu'il se rapproche ou s'éloigne de l'observateur, respectivement.

La propriété `fieldOfView` définit un angle compris entre 0 et 180 degrés qui détermine la puissance de la projection de perspective. Plus la valeur est élevée, plus forte est la distorsion appliquée à un objet d'affichage qui se déplace sur son axe z. Une valeur `fieldOfView` basse entraîne une mise à l'échelle minimale et les objets ne semblent reculer que très légèrement. Une valeur élevée entraîne une plus grande distorsion et une impression plus prononcée de mouvement. La valeur maximale, 180 degrés, produit un effet d'objectif œil-de-poisson extrême.

Centre de la projection

La propriété `projectionCenter`représente le point de fuite de la projection de perspective. Elle est appliquée comme décalage du point d'alignement par défaut (0,0) dans l'angle supérieur gauche de la scène.

A mesure qu'il semble s'éloigner de l'observateur, un objet s'incline vers le point de fuite et finit par disparaître. Imaginez un couloir extrêmement long. Lorsque vous regardez dans le couloir, les bords des murs convergent vers un point de fuite tout au fond.

Si le point de fuite se trouve au centre de la scène, le couloir disparaît vers un point central. La valeur par défaut de la propriété `projectionCenter` correspond au centre de la scène. Si, par exemple, vous souhaitez que des éléments apparaissent sur la gauche de la scène et une zone 3D sur la droite de la scène, définissez la propriété `projectionCenter` sur un point situé dans la partie droite de la scène pour en faire le point de fuite de votre zone d'affichage 3D.

Distance focale

La propriété `focalLength` représente la distance séparant l'origine du point de vue (0,0,0) de l'emplacement de l'objet d'affichage sur son axe z.

Une distance focale élevée est similaire à un téléobjectif : le champ de vision est étroit et les distances entre les objets compressées. Une distance focale courte s'assimile à un objectif à grand angle et se caractérise par un champ de vision large et une distorsion prononcée. Une distance focale moyenne correspond approximativement à ce que voit l'œil humain.

En règle générale, la propriété `focalLength` est recalculée dynamiquement pendant la transformation de perspective au fur et à mesure du déplacement de l'objet d'affichage, mais il est possible de la définir explicitement.

Valeurs par défaut de la projection de perspective

L'objet PerspectiveProjection par défaut créé sur la racine possède les valeurs suivantes :

- `fieldOfView : 55`
- `perspectiveCenter : stagewidth/2, stageHeight/2`
- `focalLength : stageWidth/ 2 * ( cos(fieldOfView/2) / sin(fieldOfView/2) )`

Ces valeurs sont appliquées si vous ne créez pas votre propre objet PerspectiveProjection.

Vous pouvez instancier votre propre objet PerspectiveProjection dans l'intention de modifier vous-même les propriétés `projectionCenter` et `fieldOfView`. Dans ce cas, les valeurs par défaut du nouvel objet sont les suivantes, en supposant que la scène mesure 500 sur 500 par défaut :

- `fieldOfView : 55`
- `perspectiveCenter : 250,250`
- `focalLength : 480.24554443359375`

Exemple : projection de perspective

L'exemple suivant illustre l'utilisation de la projection de perspective pour créer l'espace 3D. Il indique comment modifier le point de fuite et la projection de perspective de l'espace par le biais de la propriété `projectionCenter`. Cette modification force un nouveau calcul des propriétés `focalLength` et `fieldOfView`, résultant en une distorsion de l'espace 3D.

Cet exemple :

1. crée un sprite appelé `center`, un cercle avec une mire ;
2. affecte les coordonnées du sprite `center` à la propriété `projectionCenter` de la propriété `perspectiveProjection` de la propriété `transform` de la racine ;
3. ajoute des écouteurs de l'événement souris qui appellent des gestionnaires qui modifient la propriété `projectionCenter` afin qu'elle suive l'emplacement de l'objet `center` ;
4. crée quatre boîtes en accordéon qui forment les murs de l'espace en perspective.

Lorsque vous testez cet exemple, ProjectionDragger.swf, faites glisser le cercle à différents emplacements. Le point de fuite suit le cercle et figure là où vous le déposez. Observez comme les boîtes qui délimitent l'espace s'étirent et se distordent plus vous éloignez le centre de projection du centre de la scène.

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application ProjectionDragger résident dans le dossier 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;
        }
    }
}
```

Pour des projections de perspective plus complexes, utilisez la classe Matrix3D.

Transformations 3D complexes

La classe Matrix3D vous permet de transformer des points 3D dans un espace de coordonnées ou de mapper des points 3D d'un espace de coordonnées sur un autre.

Il n'est pas nécessaire de comprendre les mathématiques matricielles pour pouvoir utiliser la classe Matrix3D. Ses méthodes permettent de gérer la plupart des opérations de transformation courantes. Il est inutile de vous soucier de la définition explicite ou du calcul de la valeur de chaque élément de la matrice.

Une fois la propriété `z` d'un objet d'affichage définie sur une valeur numérique, vous pouvez récupérer la matrice de transformation de l'objet par le biais de la propriété Matrix3D de l'objet Transform de l'objet d'affichage :

```
var leafMatrix:Matrix3D = this.transform.matrix3D;
```

Les méthodes de l'objet Matrix3D vous permettent de translater, faire pivoter et mettre à l'échelle l'objet d'affichage, ainsi que de lui appliquer une projection de perspective.

Utilisez la classe Vector3D et ses propriétés `x`, `y` et `z` pour gérer les points 3D. Elle peut également représenter en physique un vecteur spatial, doté d'une direction et d'une magnitude. Les méthodes de la classe Vector3D vous permettent d'effectuer des calculs courants portant sur des vecteurs spatiaux : somme, produit scalaire et produit vectoriel, par exemple.

Remarque : *la classe Vector3D n'a aucun rapport avec la classe Vector d'ActionScript. La classe Vector3D contient des propriétés et des méthodes permettant de définir et de manipuler les points 3D, alors que la classe Vector prend en charge les tableaux d'objets typés.*

Création d'objets Matrix3D

Vous pouvez créer ou récupérer des objets `Matrix3D` de trois façons principales :

1. Utilisez la méthode constructeur `Matrix3D()` pour instancier une nouvelle matrice. Le constructeur `Matrix3D()` gère un objet `Vector` contenant 16 valeurs numériques et place chacune d'elles dans une cellule de la matrice. Par exemple :

   ```
   var rotateMatrix:Matrix3D = new Matrix3D(1,0,0,1, 0,1,0,1, 0,0,1,1, 0,0,0,1);
   ```

2. Définissez la valeur de la propriété `z` d'un objet d'affichage. Récupérez ensuite la matrice de transformation de la propriété `transform.matrix3D` de cet objet.
3. Récupérez l'objet Matrix3D qui régit l'affichage des objets 3D sur la scène en extrayant la valeur de la propriété `perspectiveProjection.matrix3D` de l'objet d'affichage racine.

Application de plusieurs transformations 3D

Vous pouvez appliquer simultanément de nombreuses transformations 3D à l'aide d'un objet Matrix3D. Ainsi, pour faire pivoter, mettre à l'échelle, puis déplacer un cube, vous pourriez appliquer trois transformations distinctes à chacun de ses points. Il est cependant beaucoup plus efficace de précalculer plusieurs transformations dans un même objet Matrix3D et d'appliquer une transformation matricielle unique à chacun des points.

Remarque : *l'ordre d'application des transformations matricielles est important. Les calculs matriciels ne sont pas réversibles. Ainsi, l'application d'une rotation puis d'une translation ne donne pas le même résultat que l'opération inverse.*

L'exemple suivant illustre deux façons d'appliquer plusieurs transformations 3D.

```
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;
        }
    }
}
```

Dans la méthode `doTransforms()`, le premier bloc de code utilise les propriétés DisplayObject pour modifier la rotation, la mise à l'échelle et la position d'un rectangle. Le second bloc utilise les méthodes de la classe Matrix3D pour effectuer des transformations identiques.

L'utilisation des méthodes `Matrix3D` présente un avantage principal : tous les calculs sont déjà effectués dans la matrice. Ils sont ensuite appliqués une seule fois à l'objet d'affichage, lors de la définition de sa propriété `transform.matrix3D`. La définition des propriétés DisplayObject améliore quelque peu la lisibilité du code source. Cependant, chaque définition d'une propriété de rotation ou de mise à l'échelle donne lieu à plusieurs calculs et entraîne la modification de plusieurs propriétés de l'objet d'affichage.

Si votre code applique plusieurs fois des transformations complexes identiques à des objets d'affichage, enregistrez l'objet Matrix3D en tant que variable, puis réappliquez-le autant de fois que nécessaire.

Réorganisation de l'affichage à l'aide d'objets Matrix3D

Comme indiqué plus haut, l'ordre d'apparition des objets d'affichage dans la liste d'affichage détermine l'ordre d'apparition à l'affichage, quels que soient leurs axes z relatifs. Si votre animation transforme les propriétés d'objets d'affichage dans un ordre différent de celui de la liste d'affichage, l'ordre d'apparition des objets d'affichage ne correspondra peut-être pas à celui des axes z. Un objet qui devrait sembler plus éloigné de l'observateur risque donc d'apparaître devant un objet plus proche.

Pour avoir la certitude que l'ordre d'apparition des objets d'affichage 3D correspond à leur profondeur relative, procédez comme suit :

1. A l'aide de la méthode `getRelativeMatrix3D()` de l'objet Transform, extrayez la profondeur relative (`z-axes`) des objets d'affichage 3D enfant.
2. Supprimez les objets de la liste d'affichage à l'aide de la méthode `removeChild()`.
3. Triez les objets d'affichage en fonction de leurs valeurs d'axe z relatives.

4 Ajoutez à nouveau les enfants à la liste d'affichage en ordre inverse, par le biais de la méthode `addChild()`.

Cette réorganisation garantit que vos objets s'affichent conformément à leurs axes z relatifs.

Le code suivant garantit l'affichage correct d'une boîte 3D à six faces. Il réorganise les faces de la boîte une fois que des rotations lui ont été appliquées.

```
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);
    }
}
```

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application résident dans le dossier Samples/ReorderByZ.

Création d'effets 3D à l'aide de triangles

Dans ActionScript, vous effectuez des transformations de bitmap à l'aide de la méthode `Graphics.drawTriangles()`, car les modèles 3D sont représentés par un ensemble de triangles dans l'espace. Flash Player et AIR ne prennent néanmoins pas en charge une mémoire tampon de profondeur, car les objets d'affichage sont toujours essentiellement plats, ou 2D. (voir « Description des fonctions 3D de Flash Player et du moteur d'exécution AIR » à la page 517). La méthode `Graphics.drawTriangles()` est similaire à la méthode `Graphics.drawPath()`, en ce qu'elle accepte un ensemble de coordonnées pour tracer un tracé triangulaire.

Pour vous familiariser avec l'utilisation de `Graphics.drawPath()`, consultez la section « Tracés de dessin » à la page 342.

La méthode `Graphics.drawTriangles()` utilise une propriété Vector.<Number> pour spécifier l'emplacement des points sur le tracé triangulaire :

```
drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null, uvtData:Vector.<Number>
= null, culling:String = "none"):void
```

Le premier paramètre de `drawTriangles()`, `vertices`, est le seul paramètre obligatoire. C'est un vecteur de nombres définissant les coordonnées par lesquelles vos triangles sont tracés. Trois ensembles de coordonnées (six nombres) représentent un tracé triangulaire. Sans le paramètre `indices`, la longueur du vecteur doit systématiquement être un facteur de six, car chaque triangle nécessite trois paires de coordonnées (trois ensembles de deux valeurs x/y). Par exemple :

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([
        10,10,  100,10,  10,100,
        110,10, 110,100, 20,100]));
```

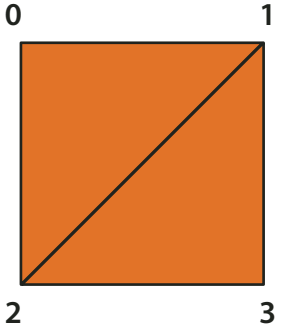
Ces triangles n'ont pas de points en commun, mais si tel était le cas, le second paramètre de `drawTriangles()`, `indices`, permettrait de réutiliser des valeurs du vecteur `vertices` pour plusieurs triangles.

Lorsque vous utilisez le paramètre `indices`, gardez à l'esprit le fait que les valeurs `indices` représentent des index de point, pas des index en rapport direct avec les éléments du tableau `vertices`. En d'autres termes, un index du vecteur `vertices` tel qu'il est défini par `indices` correspond en fait à l'index réel divisé par 2. Pour le troisième point d'un vecteur `vertices`, par exemple, utilisez une valeur `indices` de 2, même si la première valeur numérique de ce point commence à l'index vectoriel 4.

Par exemple, fusionnez deux triangles de sorte qu'ils aient en commun le bord diagonal, par le biais du paramètre `indices`:

```
graphics.beginFill(0xFF8000);
graphics.drawTriangles(
    Vector.<Number>([10,10, 100,10, 10,100, 100,100]),
    Vector.<int>([0,1,2, 1,3,2]));
```

Vous remarquerez que, bien qu'un carré résulte du tracé de deux triangles, seuls quatre points ont été spécifiés dans le vecteur `vertices`. Grâce à `indices`, les deux points partagés par les deux triangles sont réutilisés pour chacun d'eux. Le nombre total de sommets passe donc de 6 (12 nombres) à 4 (8 nombres).



Carré tracé à l'aide de deux triangles à l'aide du paramètre vertices

Cette technique s'avère utile pour les maillages triangulaires plus grands, dans lesquels la plupart des points sont partagés par plusieurs triangles.

Il est possible d'appliquer tous les remplissages aux triangles. Ils sont appliqués au maillage triangulaire résultant comme ils le seraient à toute autre forme.

Transformation d'images bitmap

Les transformations de bitmap donnent une illusion de perspective ou « texture » à un objet en trois dimensions. Vous pouvez notamment distordre une bitmap en direction d'un point de fuite, afin que l'image semble diminuer à mesure qu'elle se rapproche de celui-ci. Vous pouvez aussi utiliser une bitmap en deux dimensions pour créer une surface sur un objet en trois dimensions, donnant ainsi l'impression qu'il possède une texture ou « enveloppe ».

Surface en deux dimensions utilisant un point de fuite et objet en trois dimensions enveloppé dans une bitmap

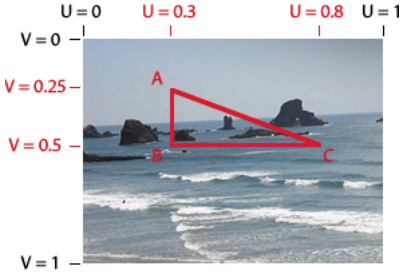
Mappage des coordonnées UV

Lorsque vous commencerez à manipuler les textures, vous souhaiterez utiliser le paramètre uvtData de `drawTriangles()`. Ce paramètre vous permet de définir le mappage des coordonnées UV pour les remplissages de bitmap.

Le mappage des coordonnées UV est une méthode d'application d'une texture à des objets. Il repose sur deux valeurs, une valeur U horizontale (x) et une valeur V verticale (y). Ces valeurs sont basées sur des pourcentages et non sur des valeurs de pixels. 0 U et 0 V correspondent au coin supérieur gauche d'une image, 1 U et 1 V à son coin inférieur droit :

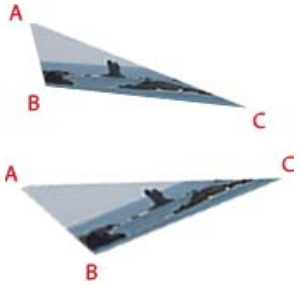
Emplacements UV 0 et 1 sur une image bitmap

Il est possible d'affecter des coordonnées UV aux vecteurs d'un triangle de sorte qu'ils s'associent aux emplacements respectifs sur une image :



Coordonnées UV d'une zone triangulaire sur une image bitmap

Les valeurs UV restent en phase avec les points du triangle.



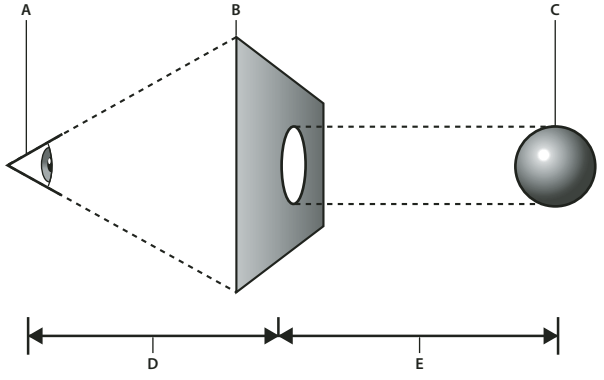
Les sommets du triangle se déplacent et l'image bitmap se distord pour que les valeurs UV d'un point individuel restent identiques.

Au fur et à mesure que des transformations 3D ActionScript sont appliquées au triangle associé à la bitmap, celle-ci est appliquée au triangle en fonction des valeurs UV. Par conséquent, au lieu d'utiliser des calculs matriciels, définissez ou réglez les valeurs UV pour créer un effet tridimensionnel.

La méthode `Graphics.drawTriangles()` accepte également une information facultative pour les transformations tridimensionnelles : la valeur T. La valeur T de uvtData représente la perspective 3D ou, plus spécifiquement, le facteur d'échelle du sommet associé. Le mappage des coordonnées UVT ajoute une correction de perspective au mappage des coordonnées UV. Par exemple, si un objet de l'espace 3D est éloigné du point de vue de telle sorte qu'il semble mesurer 50 % de sa taille « d'origine », sa valeur T correspond à 0,5. Comme les objets de l'espace 3D sont représentés à l'aide de triangles, l'emplacement de ceux-ci le long de l'axe z détermine leur valeur T. L'équation qui représente la valeur T est la suivante :

```
T = focalLength/(focalLength + z);
```

Dans cette équation, focalLength représente une distance focale ou un emplacement à l'« écran » calculé qui détermine la quantité de perspective de l'affichage.



Distance focale et valeur z
***A.** point de vue **B.** écran **C.** objet 3D **D.** valeur focalLength **E.** valeur z*

La valeur T permet de mettre à l'échelle des formes simples pour donner l'impression qu'elles se trouvent au loin. C'est généralement la valeur utilisée pour convertir les points 3D en points 2D. Dans le cas des données UVT, elle permet aussi de mettre à l'échelle une bitmap entre les points d'un triangle avec perspective.

Lorsque vous définissez des valeurs UVT, la valeur T suit directement les valeurs UV définies pour un sommet. Avec l'inclusion de T, chaque trio de valeurs du paramètre `uvtData` (U, V et T) correspond à chaque paire de valeurs du paramètre `vertices` (x et y). Avec les valeurs UV seules, uvtData.length == vertices.length. Avec l'inclusion d'une valeur T, uvtData.length = 1,5*vertices.length.

L'exemple suivant illustre un plan qui pivote, par le biais de données UVT, dans un espace 3D. Il utilise l'image ocean.jpg et une classe « d'interaction », ImageLoader, qui charge l'image afin qu'il soit possible de l'affecter à l'objet BitmapData.

La source de la classe ImageLoader (enregistrez ce code dans le fichier ImageLoader.as) se présente comme suit :

```
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);
    }
    }
}
```

Le code ActionScript qui utilise des triangles, le mappage des coordonnées UV et des valeurs T pour que l'image semble pivoter et diminuer au fur et à mesure qu'elle se rapproche d'un point de fuite est indiqué ci-dessous. Enregistrez-le dans un fichier que vous nommerez Spinning3dOcean.as :

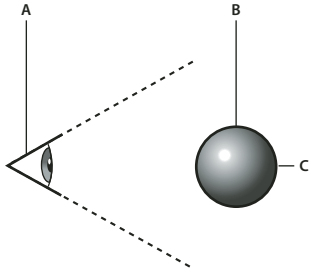
```
package {
    import flash.display.*
    import flash.events.*;
    import flash.utils.getTimer;

    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);
        }
    }
}
```

Pour tester cet exemple, enregistrez ces deux fichiers de classe dans le même répertoire qu'une image nommée « ocean.jpg ». Vous pouvez constater la transformation appliquée à la bitmap d'origine pour qu'elle semble disparaître au loin et pivoter dans l'espace 3D.

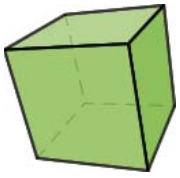
Culling

Le culling est un processus qui détermine quelles surfaces d'un objet en trois dimensions ne doivent pas être rendues par le moteur de rendu car elles ne sont pas visibles à partir du point de vue actuel. Dans l'espace 3D, la surface « arrière » d'un objet en trois dimensions n'est pas visible à partir du point de vue.



L'arrière d'un objet 3D n'est pas visible à partir du point de vue.
A. *point de vue* ***B.*** *objet 3D* ***C.*** *arrière d'un objet en trois dimensions*

Par essence, tous les triangles sont systématiquement rendus, quelles que soient leur taille, forme et position. Le culling (élimination) garantit que Flash Player ou AIR rend votre objet 3D correctement. En outre, pour éviter les cycles de rendu superflus, il est parfois souhaitable que le moteur de rendu omette certains triangles. Soit un cube en rotation dans l'espace. A tout moment donné, vous ne voyez jamais plus de trois de ses faces car celles qui ne sont pas visibles font face à l'autre direction, de l'autre côté du cube. Comme ces faces ne sont pas visibles, il est inutile que le moteur de rendu les trace. Sans élimination (culling), Flash Player ou AIR rend les faces avant et arrière.



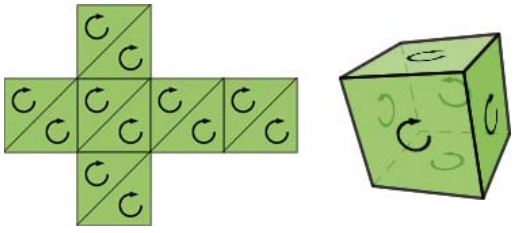
Certaines faces d'un cube ne sont pas visibles à partir du point de vue actuel.

C'est pourquoi la méthode `Graphics.drawTriangles()` gère un quatrième paramètre qui permet de définir une valeur de culling :

```
public function drawTriangles(vertices:Vector.<Number>, indices:Vector.<int> = null,
uvtData:Vector.<Number> = null, culling:String = "none"):void
```

Le paramètre culling correspond à une valeur de la classe d'énumération `TriangleCulling` : `TriangleCulling.NONE`, `TriangleCulling.POSITIVE` et `TriangleCulling.NEGATIVE`. Ces valeurs sont fonction de la direction du tracé triangulaire définissant la surface de l'objet. L'API ActionScript qui permet de déterminer le culling considère comme acquis que tous les triangles orientés vers l'extérieur d'une forme 3D sont tracés dans le même sens. Le retournement d'une face de triangle entraîne un changement de direction du tracé. A ce point, il est possible de supprimer le triangle du rendu.

Si la valeur de `TriangleCulling` est définie sur `POSITIVE`, les triangles dont le tracé a une direction positive (sens horaire) sont supprimés. Si la valeur de `TriangleCulling` est définie sur `NEGATIVE`, les triangles dont le tracé a une direction négative (sens antihoraire) sont supprimés. Dans le cas d'un cube, les surfaces orientées vers l'avant ont un tracé à la direction positive et les surfaces orientées vers l'arrière, un tracé à la direction négative.



Cube « déroulé » illustrant le sens du tracé. Lorsque le cube est « enroulé », le sens du tracé de la face arrière est inversé.

Pour comprendre le fonctionnement du processus d'élimination (culling), reprenez l'exemple de la section « Mappage des coordonnées UV » à la page 530 et définissez le paramètre de culling de la méthode `drawTriangles()` sur `TriangleCulling.NEGATIVE` :

```
container.graphics.drawTriangles(vertices, indices, uvtData, TriangleCulling.NEGATIVE);
```

Vous pouvez constater que la face « arrière » de l'image n'est pas rendue lorsque l'objet pivote.

Chapitre 24 : Utilisation de la vidéo

La vidéo avec Flash est l'une des technologies dominantes d'Internet. Toutefois, l'interface traditionnelle de la vidéo, dans un écran rectangulaire avec une barre de progression surmontant des boutons de contrôle, n'est que l'un des usages possibles de la vidéo. En ActionScript, il est possible de contrôler avec précision le chargement, la présentation et la lecture d'une vidéo.

Principes de base de la vidéo

Introduction à l'utilisation de la vidéo

La possibilité de lire et manipuler des informations vidéo en ActionScript, au même titre que les autres éléments multimédia (images, texte, animations, etc.) est l'une des principales caractéristiques d'Adobe® Flash® Player et d'Adobe® AIR™.

Lorsque vous créez un fichier vidéo Flash (FLV) dans Adobe Flash CS4 Professional, vous avez la possibilité de sélectionner un habillage, ou « enveloppe », comportant les commandes de lecture courantes. Toutefois, vous n'êtes pas limité aux options disponibles. ActionScript offre un contrôle précis du chargement, de l'affichage et de la lecture de la vidéo, et vous pouvez créer votre propre enveloppe ou utiliser une vidéo de façon beaucoup moins traditionnelle.

La gestion de la vidéo en ActionScript nécessite de travailler avec une combinaison de plusieurs classes :

- Classe Video : le contenu vidéo affiché sur la scène est une occurrence de la classe Video. La classe Video est un objet d'affichage, il est donc possible de le manipuler à l'aide des mêmes techniques que les autres objets d'affichage (positionnement, application de transformations, de filtres et de modes de fusion, etc.)
- Classe NetStream : lorsque vous chargez un fichier vidéo qui doit être contrôlé en ActionScript, une occurrence de NetStream représente la source du contenu vidéo (dans ce cas précis, un flux de données vidéo). L'utilisation d'une occurrence de NetStream nécessite d'utiliser également un objet NetConnection, qui assure la connexion avec le fichier vidéo, comme un tunnel qu'emprunteraient les données vidéo.
- Classe Camera : si vous devez gérer des données provenant d'une caméra connectée à l'ordinateur de l'utilisateur, une occurrence de Camera représente la source du contenu vidéo (la caméra de l'utilisateur et les données vidéo qu'elle transmet).

Pour charger un fichier vidéo externe, vous pouvez charger ce fichier à partir d'un serveur Web standard (téléchargement progressif) ou gérer de la vidéo en flux continu transmise par un serveur spécialisé tel que Flash® Media Server d'Adobe.

Tâches courantes d'utilisation de la vidéo

Voici quelques tâches courantes relatives à la vidéo présentées dans ce chapitre :

- Affichage et contrôle d'une vidéo
- Chargement de fichiers vidéo externes
- Contrôle de la lecture de la vidéo
- Utilisation du mode plein écran
- Gestion des métadonnées et des points de repère dans un fichier vidéo

- Capture et affichage d'un signal vidéo provenant de la caméra de l'utilisateur

Concepts importants et terminologie

- Point de repère : marqueur qu'il est possible de placer en un point spécifique d'un fichier vidéo, notamment pour l'utiliser comme signet pour repérer ce point à partir du début de la vidéo ou pour fournir des données supplémentaires associées à ce moment de la vidéo.
- Encodage : processus de conversion de données vidéo d'un format dans un autre format vidéo. Par exemple, la conversion d'une source vidéo en haute résolution dans un format plus adapté à la diffusion sur Internet.
- Image : élément de base des informations vidéo. Chaque image est une image fixe identique à un cliché photographique représentant un moment précis. La lecture en séquence à vitesse élevée de ces images fixes donne l'illusion du mouvement.
- Image-clé : image vidéo qui contient l'ensemble des informations de l'image. Les autres images qui suivent une image-clé ne contiennent que les informations décrivant leurs différences par rapport à l'image-clé, et non pas les informations d'image complètes.
- Métadonnées : informations sur un fichier vidéo intégrées à ce fichier et lues après son chargement.
- Téléchargement progressif : lorsqu'un fichier vidéo est transmis par un serveur Web standard, les données vidéo sont chargées en mode progressif, c'est-à-dire en séquences. L'avantage est qu'il est possible de commencer à diffuser la vidéo avant la fin du téléchargement complet. Toutefois, il est alors impossible de passer directement à une partie de la vidéo qui n'a pas encore été chargée.
- Flux continu : pour éviter le téléchargement progressif, il est possible d'utiliser un serveur vidéo spécial pour diffuser de la vidéo sur Internet selon une technique connue sous le nom de flux continu. Avec la diffusion en flux continu, l'ordinateur client ne charge jamais toute la vidéo à la fois. Pour accélérer les délais de chargement, l'ordinateur n'a besoin, à un moment donné, que d'une partie de l'ensemble des informations vidéo. Comme un serveur spécial contrôle la diffusion du contenu vidéo, une partie quelconque de celle-ci peut être transmise à tout moment, et il n'est donc pas nécessaire d'attendre qu'elle soit chargée pour y accéder.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester des exemples de code. Etant donné que ce chapitre traite de l'utilisation de la vidéo dans ActionScript, un grand nombre d'exemples de code de ce chapitre impliquent l'utilisation d'un objet vidéo (créé et placé sur la scène dans l'outil de programmation Flash, ou créé à l'aide d'ActionScript). Pour tester un exemple, il est nécessaire d'afficher le résultat dans Flash Player ou AIR afin de voir l'effet du code sur la vidéo.

La plupart des exemples de code manipulent un objet Video sans créer l'objet de façon explicite. Pour tester les codes de ce chapitre :

1. Créez un document Flash vide.
2. Sélectionnez une image-clé dans le scénario.
3. Ouvrez le panneau Actions et copiez le code dans le panneau Script.
4. Si nécessaire, ouvrez le panneau Bibliothèque.
5. Depuis le menu du panneau Bibliothèque, choisissez Nouvelle Vidéo.
6. Dans la boîte de dialogue Propriétés de la vidéo, entrez un nom pour le nouveau symbole vidéo, puis sélectionnez Vidéo (contrôlée par ActionScript) dans le champ Type. Cliquez sur OK pour créer le symbole vidéo.
7. Faites glisser une occurrence de votre symbole vidéo du panneau Bibliothèque vers la Scène.

8 Sélectionnez l'occurrence de la vidéo, puis donnez-lui un nom dans l'Inspecteur des propriétés. Le nom doit correspondre au nom utilisé pour l'occurrence de l'objet Video dans l'exemple de code (par exemple, si le code manipule un objet Video appelé `vid`, vous devez également appeler votre occurrence de la scène `vid`).

9 Exécutez le programme en choisissant Contrôle > Tester l'animation.

Les résultats du code manipulant la vidéo s'affichent à l'écran, comme indiqué dans le code.

Outre l'exemple de code à proprement parler, certains exemples de code de ce chapitre comportent une définition de classe. Dans ces exemples de code, outre les étapes précédentes, et avant de tester le fichier SWF, vous devrez créer la classe utilisée dans l'exemple. Pour créer une classe définie dans un exemple de code :

1 Assurez-vous d'avoir enregistré le fichier FLA qui sera utilisé lors du test.

2 Dans le menu principal, choisissez Fichier > Nouveau.

3 Dans la boîte de dialogue Nouveau document, dans la section Type, choisissez Fichier ActionScript. Cliquez sur OK pour créer le nouveau fichier ActionScript.

4 Copiez le code de définition de la classe de l'exemple dans le document ActionScript.

5 Dans le menu principal, choisissez Fichier > Enregistrer. Enregistrez le fichier dans le même répertoire que le document Flash. Le nom du fichier doit correspondre au nom de la classe du code. Par exemple, si le code définit une classe appelée « Video Test », enregistrez le fichier ActionScript sous la forme « VideoTest.as ».

6 Revenez au document Flash.

7 Exécutez le programme en choisissant Contrôle > Tester l'animation.

Les résultats de l'exemple s'affichent à l'écran.

D'autres techniques de test d'exemples de code sont expliquées plus en détail à la section « Test des exemples de code contenus dans un chapitre » à la page 36.

Présentation des formats vidéo

Outre le format vidéo Adobe FLV, Flash Player et Adobe AIR prennent en charge les contenus vidéo et audio codés au format H.264 et HE-AAC des formats de fichier standard MPEG-4. Ces formats diffusent des vidéos de qualité supérieure à des débits inférieurs. Les développeurs peuvent utiliser les outils standard, notamment Adobe Premiere Pro et Adobe After Effects, pour créer et présenter du contenu vidéo de grande qualité.

Type	Format	Contenant
Vidéo	H.264	MPEG-4 : MP4, M4V, F4V, 3GPP
Vidéo	Fichier FLV	Sorenson Spark
Vidéo	Fichier FLV	ON2 VP6
Audio	AAC+ / HE-AAC / AAC v1 / AAC v2	MPEG-4:MP4, M4V, F4V, 3GPP
Audio	Mp3	Mp3
Audio	Nellymoser	Fichier FLV
Audio	Speex	Fichier FLV

Compatibilité de Flash Player et AIR avec les fichiers vidéo codés

Flash Player 7 prend en charge les fichiers FLV codés avec le codec vidéo Sorenson™ Spark™. Flash Player 8 prend en charge les fichiers FLV codés avec l'encodeur Sorenson ou On2 VP6 dans Flash Professional 8. Le codec On2 VP6 prend en charge un canal alpha.

Flash Player 9.0.115.0 et les versions ultérieures prennent en charge les fichiers dérivés du format conteneur standard MPEG-4. Ces fichiers sont les suivants : F4V, MP4, M4A, MOV, MP4V, 3GP et 3G2, s'ils contiennent de la vidéo H.264 ou de l'audio codé au format HE-AAC v2, ou les deux. H.264 produit une qualité vidéo supérieure à un débit inférieur par rapport au même profil d'encodage dans Sorenson ou On2. HE-AAC v2 est une extension du format AAC, format audio standard défini dans la norme vidéo MPEG-4. HE-AAC v2 utilise les techniques de réplique spectrale de bande (SBR - Spectral Band Replication) et de stéréo paramétrique pour optimiser l'efficacité de l'encodage à des vitesses de transfert inférieures.

Le tableau suivant répertorie les codecs pris en charge. Il décrit également le format de fichier SWF correspondant et les versions de Flash Player et d'AIR requises pour les lire :

Codec	**Version du format de fichier SWF (version de publication la plus récente prise en charge)**	**Flash Player et AIR (version la plus récente requise pour la lecture)**
Sorenson Spark	6	Flash Player 6, Flash Lite 3
On2 VP6	6	Flash Player 8, Flash Lite 3. Seuls Flash Player 8 et les versions ultérieures prennent en charge la publication et la lecture des vidéos On2 VP6.
H.264 (MPEG-4 Part 10)	9	Flash Player 9 Mise à jour 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 Mise à jour 3, AIR 1.0
Speex (audio)	10	Flash Player 10, AIR 1.5
Nellymoser	6	Flash Player 6

Présentation des formats de fichiers vidéo Adobe F4V et FLV

Adobe fournit des formats de fichiers vidéo pour diffuser en continu un contenu à Flash Player et à AIR. Pour une description complète de ces formats de fichiers vidéo, consultez www.adobe.com/go/video_file_format_fr.

Format de fichier vidéo F4V

A partir de Flash Player Update 3 (9.0.115.0) et AIR 1.0, Flash Player et AIR prennent en charge le format vidéo Adobe F4V qui découle du format ISO MP4. Les sous-ensembles du format prennent en charge des fonctions diverses. Flash Player s'attend à ce qu'un fichier F4V valide commence par l'une des boîtes de haut niveau suivantes :

- ftyp

  La boîte ftyp identifie les fonctions qu'un programme doit prendre en charge pour diffuser un format de fichier particulier.

- moov

  La boîte moov est effectivement l'en-tête d'un fichier F4V. Elle contient une ou plusieurs autres boîtes qui à leur tour contiennent d'autres boîtes et qui définissent la structure des données F4V. Un fichier F4V ne doit contenir qu'une seule boîte moov.

- mdat

  Une boîte mdat contient les données " utiles " pour le fichier F4V. Un fichier F4V ne doit contenir qu'une seule boîte mdat. Une boîte moov doit également se trouver dans le fichier parce que l boîte mdat ne peut pas être comprise si elle est seule.

Les fichiers F4V prennent en charge des entiers multioctets dans un ordre d'octets gros-boutiste, dans lequel l'octet le plus significatif paraît le premier, à l'adresse la plus basse.

Format de fichier vidéo FLV

Le format de fichier Adobe FLV contient des données audio et vidéo codées pour un acheminement via Flash Player. Vous pouvez utiliser un codeur, tel que Adobe Media Encoder ou Sorenson™ Squeeze, pour convertir un fichier vidéo QuickTime ou Windows Media en un fichier FLV.

Remarque : *vous pouvez créer des fichiers FLV en important la vidéo dans Flash et en l'exportant sous forme de fichier FLV. Vous pouvez utiliser le module d'exportation FLV pour exporter des fichiers FLV à partir des applications de montage vidéo prises en charge. Pour charger des fichiers FLV à partir d'un serveur Web, enregistrez l'extension de fichier et le type MIME auprès de votre serveur Web. Pour ce faire, consultez la documentation du serveur. Le type MIME des fichiers FLV est* `video/x-flv`*. Pour plus d'informations, consultez la section « A propos de la configuration de fichier FLV pour l'hébergement sur un serveur » à la page 570.*

Pour plus d'informations sur les fichiers FLV, consultez la section « Rubriques avancées pour les fichiers FLV » à la page 570.

Vidéo externe contre vidéo intégrée

L'utilisation des fichiers vidéo externes offre certaines fonctionnalités qui ne sont pas disponibles avec l'utilisation de la vidéo importée :

- Les clips vidéo de longue durée peuvent être utilisés dans votre application sans ralentir la lecture. Les fichiers vidéo externes utilisent la mémoire cache, ce qui signifie que les fichiers volumineux sont enregistrés en petites parties et sont accessibles dynamiquement. Par conséquent, les fichiers F4V et FLV externes ne nécessitent pas autant de mémoire que les fichiers vidéo intégrés.
- Un fichier vidéo externe peut avoir une cadence différente de celle du fichier SWF dans lequel il est lu. Par exemple, vous pouvez définir la cadence du fichier SWF sur 30i/s (images par seconde) et celle de l'image vidéo sur 21i/s. Ce réglage vous offre un meilleur contrôle de la vidéo que la vidéo intégrée, pour assurer une lecture vidéo fluide. Il permet aussi de lire les fichiers vidéo à différentes cadences d'images sans avoir à altérer un contenu SWF existant.
- Avec les fichiers vidéo externes, la lecture du contenu SWF n'est pas interrompue lors du chargement du fichier vidéo. Les fichiers vidéo importés peuvent parfois interrompre la lecture du document pour exécuter certaines fonctions, par exemple pour accéder à un lecteur de CD-ROM. Les fichiers vidéo peuvent exécuter des fonctions indépendamment du contenu SWF, sans interrompre la lecture.
- Le sous-titrage du contenu vidéo est plus facile avec les fichiers FLV externes, car les gestionnaires d'événements permettent d'accéder aux métadonnées de la vidéo.

Présentation de la classe Video

La classe Video permet d'afficher un flux vidéo en direct dans une application sans l'imbriquer dans votre fichier SWF. Vous pouvez capturer et lire du contenu vidéo en direct à l'aide de la méthode `Camera.getCamera()`. Vous pouvez également utiliser la classe Video pour lire les fichiers vidéo en HTTP ou sur le système de fichiers local. Il existe plusieurs façons d'utiliser la classe Video dans vos projets.

- Chargement dynamique d'un fichier vidéo à l'aide des classes NetConnection et NetStream, et affichage de la vidéo dans un objet Video.
- Capture du signal provenant de la caméra de l'utilisateur. Pour plus d'informations, consultez la section « Capture d'un signal vidéo provenant de la caméra de l'utilisateur » à la page 563.
- Utilisation du composant FLVPlayback.

Remarque : *les occurrences d'un objet Video sur la scène sont des occurrences de la classe Video.*

Bien que la classe Video se trouve dans le package flash.media, elle hérite de la classe flash.display.DisplayObject. Par conséquent, toutes les fonctionnalités des objets d'affichage (transformation de matrice et filtres par exemple) s'appliquent aussi aux occurrences de l'objet Video.

Pour plus d'informations, consultez la section « Manipulation des objets d'affichage » à la page 299, ainsi que les chapitres « Utilisation de la géométrie » à la page 350 et « Filtrage des objets d'affichage » à la page 363.

Chargement de fichiers vidéo

Le chargement de fichiers vidéo à l'aide des classes NetStream et NetConnection s'effectue en plusieurs étapes.

1. Créez un objet NetConnection. Dans le cas d'une connexion à un fichier vidéo local ou à un fichier qui n'utilise pas de serveur, tel que le serveur Flash Media Server 2 d'Adobe, transmettez `null` à la méthode `connect()` pour lire les fichiers vidéo depuis une adresse HTTP ou un lecteur local. Dans le cas d'une connexion à un serveur, définissez le paramètre sur l'URI de l'application qui contient le fichier vidéo sur le serveur.

```
var nc:NetConnection = new NetConnection();
nc.connect(null);
```

2. Créez un objet NetStream qui prend un objet NetConnection comme paramètre, puis spécifiez le fichier vidéo que vous souhaitez charger. Le fragment de code ci-dessous connecte un objet NetStream à l'occurrence de NetConnection spécifiée et charge un fichier vidéo nommé video.mp4 dans le même répertoire que le fichier SWF :

```
var ns:NetStream = new NetStream(nc);
ns.addEventListener(AsyncErrorEvent.ASYNC_ERROR, asyncErrorHandler);
ns.play("video.mp4");
function asyncErrorHandler(event:AsyncErrorEvent):void
{
    // ignore error
}
```

3. Créez un objet Video et affectez-lui l'objet NetStream précédemment créé à l'aide de la méthode `attachNetStream()` de la classe Video. Vous pouvez ensuite ajouter l'objet vidéo à la liste d'affichage à l'aide de la méthode `addChild()`, comme dans l'exemple ci-dessous :

```
var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

Etant donné que Flash Player exécute ce code, il tente de charger le fichier vidéo video.mp4 depuis le même répertoire que votre fichier SWF.

Contrôle de la lecture de la vidéo

La classe NetStream comporte quatre méthodes principales pour contrôler la lecture vidéo :

`pause()` : interrompt la lecture d'un flux vidéo. Si la lecture de la vidéo est déjà en pause, l'appel de cette méthode n'a aucun effet.

resume() : reprend la lecture d'un flux vidéo en pause. Si la vidéo est en cours de lecture, l'appel de cette méthode n'a aucun effet.

seek() : recherche l'image-clé la plus proche de l'emplacement spécifié (décalage, exprimé en secondes, par rapport au début du flux continu).

togglePause() : interrompt ou reprend la lecture d'un flux continu.

Remarque : *il n'existe pas de méthode `stop()`. Pour arrêter la lecture de la vidéo, il est nécessaire de la mettre en pause et de retourner au début du flux vidéo.*

Remarque : *la méthode `play()` ne reprend pas la lecture, elle est destinée au chargement de fichiers vidéo.*

L'exemple suivant montre comment contrôler la lecture d'une vidéo à l'aide de divers boutons. Pour exécuter cet exemple, créez un document et ajoutez quatre occurrences de boutons à l'espace de travail (`pauseBtn`, `playBtn`, `stopBtn` et `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();
}
```

Un clic sur l'occurrence de bouton `pauseBtn` pendant la lecture de la vidéo provoque la mise en pause de celle-ci. Si la lecture de la vidéo est déjà en pause, l'appel de cette méthode n'a aucun effet. Un clic sur l'occurrence de `playBtn` reprend la lecture de la vidéo si celle-ci était en pause, sinon ce bouton n'a aucun effet.

Détection de la fin d'un flux vidéo

Pour afficher le début et la fin d'un flux vidéo, vous devez ajouter à l'occurrence de NetStream un écouteur pour l'événement `netStatus`. L'exemple suivant montre comment écouter les divers codes pendant la lecture de la vidéo :

```
ns.addEventListener(NetStatusEvent.NET_STATUS, statusHandler);
function statusHandler(event:NetStatusEvent):void
{
    trace(event.info.code)
}
```

Le code précédent affiche le résultat suivant :

```
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
```

Les deux codes d'événement qu'il est nécessaire d'écouter sont « NetStream.Play.Start » et « NetStream.Play.Stop », qui signalent le début et la fin de la lecture de la vidéo. Le fragment de code suivant utilise une instruction « switch » pour filtrer ces deux codes et émettre un message :

```
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;
    }
}
```

En écoutant l'événement `netStatus` (`NetStatusEvent.NET_STATUS`), vous pouvez créer un lecteur vidéo qui chargera la vidéo suivante dans une liste de lecture une fois la lecture de la vidéo en cours terminée.

Lecture de vidéos en mode plein écran

Flash Player et AIR vous permettent de créer une application plein écran pour la lecture de votre vidéo et prennent en charge la mise à l'échelle de la vidéo au mode plein écran.

Pour le contenu AIR s'exécutant en mode plein écran, les options économiseur d'écran et économie d'énergie du système sont désactivées lors de la lecture jusqu'à ce que le signal vidéo s'arrête ou que l'utilisateur quitte le mode plein écran.

Pour plus de détails sur l'utilisation du mode plein écran, consultez la section « Utilisation du mode plein écran » à la page 293.

Activation du mode plein écran pour Flash Player dans un navigateur

Avant que vous ne puissiez implémenter le plein écran pour Flash Player dans un navigateur, activez-le via le modèle de publication de votre application. Les modèles qui permettent le mode plein écran comprennent les balises `<object>` et `<embed>`, qui contiennent un paramètre `allowFullScreen`. L'exemple suivant affiche le paramètre `allowFullScreen` dans une balise `<embed>`.

```
<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>
```

Dans Flash, choisissez Fichier -> Paramètres de publication, puis dans la boîte de dialogue Paramètres de publication, cliquez sur l'onglet HTML, puis sélectionnez le modèle Flash seulement - Autorisation du Plein écran.

Dans Flex, assurez-vous que le modèle HTML inclue les balises `<object>` et `<embed>` qui prennent en charge le plein écran.

Activation du mode plein écran

Pour le contenu de Flash Player s'exécutant dans un navigateur, le mode plein écran de la vidéo est activé en réponse à un clic de souris ou à une pression sur une touche. Par exemple, vous pouvez activer le mode plein écran lorsque l'utilisateur clique sur un bouton appelé Plein écran ou sélectionne une commande Plein écran dans un menu contextuel. Pour répondre à l'utilisateur, ajoutez un écouteur d'événement à l'objet sur lequel se produit l'action. Le code suivant ajoute un écouteur d'événement à un bouton sur lequel l'utilisateur clique pour accéder au mode plein écran :

```
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;

}
```

Le code active le mode plein écran en définissant la propriété `Stage.displayState` sur `StageDisplayState.FULL_SCREEN`. Ce code affiche la totalité de la scène en mode plein écran et met à l'échelle la vidéo selon les proportions de l'espace qu'elle occupe sur la scène.

La propriété `fullScreenSourceRect` vous permet de spécifier une zone particulière de la scène en vue de l'afficher en mode plein écran. Définissez tout d'abord le rectangle que vous souhaitez afficher en mode plein écran. Ensuite, affectez-le à la propriété `Stage.fullScreenSourceRect`. Cette version de la fonction `fullScreenButtonHandler()` ajoute deux lignes supplémentaires de code qui n'activent le plein écran que pour la vidéo.

```
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;
}
```

Bien que cet exemple invoque un gestionnaire d'événement en réponse à un clic de souris, la technique d'activation du mode plein écran est la même pour Flash Player et pour AIR. Définissez le rectangle que vous souhaitez mettre à l'échelle, puis définissez la propriété `Stage.displayState`. Pour plus d'informations, consultez le Guide de référence du langage et des composants ActionScript 3.0.

L'exemple complet qui suit ajoute le code permettent de créer la connexion et l'objet NetStream pour la vidéo, puis commence à le lire.

```
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
        }
    }
}
```

La fonction `onMetaData()` est une fonction de rappel pour gérer les métadonnées de la vidéo, si celles-ci existent. Une fonction de rappel est une fonction que le moteur d'exécution appelle en réponse à certains types d'occurrences ou d'événements. Dans cet exemple, la fonction `onMetaData()` est une souche capable de fournir la fonction. Pour plus d'informations, consultez la section « Ecriture de méthodes de rappel pour les métadonnées et les points de repère » à la page 549

Désactivation du mode plein écran

Tout utilisateur peut désactiver le mode plein écran à l'aide de l'un des raccourcis clavier, notamment en appuyant sur la touche Echap. En ActionScript, vous pouvez désactiver le mode plein écran en définissant la propriété `Stage.diplayState` sur `StageDisplayState.NORMAL`. Dans l'exemple suivant, le code désactive le mode plein écran lorsque l'événement `netStatus` de NetStream.Play.Stop se produit.

```
videoStream.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);

private function netStatusHandler(event:NetStatusEvent)
{
    if(event.info.code == "NetStream.Play.Stop")
        stage.displayState = StageDisplayState.NORMAL;
}
```

Accélération matérielle en mode plein écran

Lorsque vous définissez la propriété `Stage.fullScreenSourceRect` pour mettre à l'échelle un segment rectangulaire de la scène au mode plein écran, Flash Player ou AIR utilise l'accélération matérielle, à condition que cette fonction soit disponible et activée. Le moteur d'exécution utilise l'adaptateur vidéo de l'ordinateur pour accélérer la mise à l'échelle de la vidéo ou d'une portion de la scène au mode plein écran.

Pour plus d'informations sur l'accélération matérielle en mode plein écran, consultez la section « Utilisation du mode plein écran » à la page 293.

Lecture de fichiers vidéo en flux continu

Pour effectuer une lecture en flux continu à partir d'un serveur Flash Media Server, vous pouvez utiliser les classes NetConnection et NetStream afin d'établir la connexion avec une occurrence de serveur distant et lire le flux spécifié. Pour spécifier un serveur RTMP (Real-Time Messaging Protocol), il suffit de transmettre l'URL RTMP désirée, par exemple « `rtmp://localhost/appName/appInstance` » à la méthode `NetConnection.connect()` au lieu de lui transmettre la valeur null. Pour lire un flux vidéo direct ou enregistré à partir du serveur Flash Media Server spécifié, passez soit un identifiant (pour le signal vidéo en direct publié par `NetStream.publish()`), soit le nom du fichier enregistré, à la méthode `NetStream.play()`. Pour plus d'informations, consultez la documentation de Flash Media Server.

Présentation des points de repère

Vous pouvez intégrer des points de repère dans un fichier vidéo F4V ou FLV durant le codage. A l'origine, les points de repère étaient intégrés dans des films pour signaler visuellement au projectionniste que la bobine s'approchait de la fin. Dans les formats vidéo Adobe F4V et FLV, un point de repère vous permet de déclencher une ou plusieurs actions dans votre application au moment où il survient dans le flux vidéo.

Vous pouvez utiliser plusieurs types de points de repère avec Flash Video. ActionScript permet d'interagir avec les points de repère que vous intégrez dans un fichier vidéo lorsque vous le créez.

- Points de repère de navigation : vous intégrez des points de repère de navigation dans le flux et le paquet de métadonnées vidéo lorsque vous codez le fichier vidéo. Les points de repère de navigation permettent aux utilisateurs de rechercher une partie spécifique d'un fichier.
- Points de repère d'événement : vous intégrez des points de repère d'événement dans le flux et le paquet de métadonnées vidéo lorsque vous codez le fichier vidéo. Il est possible d'écrire du code pour gérer les événements qui sont déclenchés aux points spécifiés pendant la lecture.
- Points de repère ActionScript : les points de repère ActionScript sont disponibles uniquement pour le composant FLVPlayback de Flash. Les points de repère ActionScript sont des points de repère externes que vous créez et auxquels vous accédez à l'aide du code ActionScript. Du code permet de déclencher ces points de repère en fonction de la lecture vidéo. Ces points de repère sont moins précis que les points de repère intégrés (jusqu'à un dixième de seconde), car le lecteur vidéo les analyse séparément. Si vous avez l'intention de créer une application dans laquelle il sera possible d'atteindre un point de repère, créez et intégrez les points de repère lors de l'encodage du fichier, au lieu d'utiliser des points de repère ActionScript. Il est préférable d'intégrer les points de repère dans le fichier FLV, car ils sont alors plus précis.

Les points de repère de navigation créent une image-clé à l'emplacement spécifié, pour permettre de déplacer la tête de lecture du lecteur vidéo à cet emplacement. Vous pouvez définir des points particuliers dans un fichier vidéo pour permettre aux utilisateurs d'atteindre un emplacement précis. Par exemple, si votre vidéo contient plusieurs chapitres et segments, vous pouvez la contrôler en intégrant des points de repère de navigation dans le fichier vidéo.

Pour plus d'informations sur le codage de fichiers vidéo Adobe avec des points de repère, consultez la section « Intégration de points de repère » du guide *Utilisation de Flash*.

Vous pouvez accéder aux paramètres des points de repère à l'aide de code ActionScript. Les paramètres de points de repère font partie de l'objet d'événement reçu du gestionnaire de rappel.

Le gestionnaire d'événement `NetStream.onCuePoint` permet de déclencher certaines actions dans votre code lorsque le fichier FLV atteint un point de repère spécifique.

Pour synchroniser une action avec un point de repère dans un fichier vidéo F4V, vous devez récupérer les données de point de repère des fonctions de rappel `onMetaData()` ou `onXMPData()` et déclencher le point de repère à l'aide de la classe Timer dans ActionScript 3.0. Pour plus d'informations sur les points de repère de F4V, consultez la section « Utilisation d'onXMPData() » à la page 560.

Pour plus d'informations sur l'utilisation des points de repère et des métadonnées, consultez la section « Ecriture de méthodes de rappel pour les métadonnées et les points de repère » à la page 549.

Ecriture de méthodes de rappel pour les métadonnées et les points de repère

Vous pouvez déclencher des actions au sein de votre application lorsque le lecteur reçoit des métadonnées spécifiques ou lorsque des points de repère particuliers sont atteints. Lorsque ces événements se produisent, vous devez utiliser des méthodes de rappel spécifiques en tant que gestionnaires d'événements. La classe NetStream spécifie les événements de métadonnées suivants qui se produisent lors de la lecture : `onCuePoint` (fichiers FLV uniquement), `onImageData`, `onMetaData`, `onPlayStatus`, `onTextData` et `onXMPData`.

Si vous ne créez pas de méthodes de rappel pour ces gestionnaires, Flash Player est susceptible de générer des erreurs. Par exemple, le code ci-dessous lit le fichier FLV video.flv situé dans le même dossier que le fichier SWF :

```
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);
```

Le code ci-dessus charge un fichier vidéo local nommé video.flv et attend la distribution de l'événement `asyncError` (`AsyncErrorEvent.ASYNC_ERROR`). Cet événement est distribué lorsqu'une exception est renvoyée par du code asynchrone natif. Dans notre cas, il est distribué lorsque le fichier vidéo contient des métadonnées ou des informations de point de repère et que les écouteurs appropriés n'ont pas été définis. Le code ci-dessus gère l'événement `asyncError` et ignore l'erreur si vous n'êtes pas intéressé par les métadonnées ou les informations de point de repère. Si vous disposiez d'un fichier FLV avec des métadonnées et plusieurs points de repère, la fonction `trace()` afficherait les messages d'erreur suivants :

```
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.
```

L'erreur est renvoyée parce que l'objet NetStream n'a pas trouvé de méthode de rappel pour `onMetaData` ou `onCuePoint`. Il existe plusieurs façons de définir ces méthodes de rappel dans une application.

Définir la propriété « client » de l'objet NetStream comme Object

En définissant la propriété `client` comme étant soit un object, soit une sous-classe de NetStream, vous pouvez re-router les méthodes de rappel de `onMetaData` et `onCuePoint` ou les ignorer totalement. L'exemple suivant montre comment utiliser un objet vide pour ignorer les méthodes de rappel sans écouter l'événement `asyncError` :

```
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);
```

Pour écouter les méthodes de rappel de `onMetaData` ou `onCuePoint`, il est nécessaire de définir des méthodes pour les gérer, comme dans le fragment de code suivant :

```
var customClient:Object = new Object();
customClient.onMetaData = metaDataHandler;
function metaDataHandler(infoObject:Object):void
{
    trace("metadata");
}
```

Le code ci-dessus écoute la méthode de rappel de `onMetaData` et appelle la méthode `metaDataHandler()`, qui renvoie une chaîne. Si Flash Player trouve un point de repère, aucune erreur n'est renvoyée bien qu'aucune méthode de rappel de `onCuePoint` ne soit définie.

Créer une classe et définir des méthodes pour gérer les méthodes de rappel

Le code suivant définit la propriété `client` de l'objet NetStream comme étant une classe personnalisée, CustomClient, qui définit des gestionnaires pour les méthodes de rappel :

```
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);
```

La classe CustomClient contient le code suivant :

```
package
{
    public class CustomClient
    {
        public function onMetaData(infoObject:Object):void
        {
            trace("metadata");
        }
    }
}
```

La classe CustomClient définit un gestionnaire pour le rappel de `onMetaData`. Si un point de repère est détecté et que le gestionnaire de rappel de `onCuePoint` est appelé, un événement `asyncError` (`AsyncErrorEvent.ASYNC_ERROR`) est distribué pour informer que flash.net.NetStream n'a pas pu appeler de rappel pour onCuePoint. Pour éviter l'apparition de cette erreur, il est nécessaire de définir soit une méthode de rappel de `onCuePoint` dans la classe CustomClient, soit un gestionnaire d'événement pour l'événement `asyncError`.

Etendre la classe NetStream et lui ajouter des méthodes pour gérer les méthodes de rappel

Le code suivant crée une occurrence de la classe CustomNetStream, qui sera définie ultérieurement :

```
var ns:CustomNetStream = new CustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

L'exemple de code suivant définit la classe CustomNetStream qui étend la classe NetStream, prend en charge la création de l'objet NetConnection nécessaire, et gère les méthodes de rappel de `onMetaData` et `onCuePoint` :

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

Si vous souhaitez renommer les méthodes `onMetaData()` et `onCuePoint()` de la classe CustomNetStream, utilisez le code suivant :

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

Etendre la classe NetStream et la rendre dynamique

Vous pouvez étendre la classe NetStream en créant une sous-classe dynamique afin de pouvoir ajouter dynamiquement les gestionnaires de rappel de `onCuePoint` et `onMetaData`. Le code suivant en fait la démonstration :

```
var ns:DynamicCustomNetStream = new DynamicCustomNetStream();
ns.play("video.flv");

var vid:Video = new Video();
vid.attachNetStream(ns);
addChild(vid);
```

La classe DynamicCustomNetStream contient le code suivant :

```
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);
        }
    }
}
```

Même sans gestionnaire pour le rappel de `onMetaData` et `onCuePoint`, aucune erreur n'est renvoyée puisque la classe DynamicCustomNetStream est dynamique. Si vous souhaitez définir des méthodes pour les gestionnaires de rappel de `onMetaData()` et `onCuePoint()`, utilisez le code suivant :

```
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");
}
```

Définir la propriété « client » de l'objet NetStream comme this

En donnant à la propriété `client` la valeur `this`, l'application recherche dans la portée des méthodes `onMetaData()` et `onCuePoint()`. Le code suivant en est un exemple :

```
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);
```

Si les gestionnaires de rappel de `onMetaData` ou `onCuePoint` sont appelés et qu'il n'existe pas de méthode pour gérer le rappel, aucune erreur n'est générée. Pour gérer ces gestionnaires de rappel, créez des méthodes `onMetaData()` et `onCuePoint()` dans votre code, comme dans le fragment de code suivant :

```
function onMetaData(infoObject:Object):void
{
    trace("metadata");
}
function onCuePoint(infoObject:Object):void
{
    trace("cue point");
}
```

Utilisation des points de repère et des métadonnées

Les méthodes de rappel NetStream vous permettent de capturer et de traiter les événements de point de repère et de métadonnées lorsque la vidéo est en cours de lecture.

Utilisation des points de repère

Le tableau ci-dessous décrit les méthodes de rappel que vous pouvez utiliser pour capturer les points de repère F4V et FLV dans Flash Player et AIR.

Moteur d'exécution	F4V	FLV
Flash Player 9/ AIR1.0		OnCuePoint
		OnMetaData
Flash Player 10		OnCuePoint
	OnMetaData	OnMetaData
	OnXMPData	OnXMPData

L'exemple suivant utilise une boucle `for..in` simple pour effectuer une itération sur chaque propriété du paramètre `infoObject` que reçoit la fonction `onCuePoint()`. Il appelle à fonction trace() pour afficher un message lors de la réception de données de point de repère :

```
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]);
    }
}
```

Le résultat suivant apparaît :

```
parameters:
name: point1
time: 0.418
type: navigation
```

Ce code utilise l'une des techniques disponibles pour définir l'objet pour lequel la méthode de rappel est exécutée. Vous pouvez utiliser d'autres techniques ; pour plus d'informations, consultez la section « Ecriture de méthodes de rappel pour les métadonnées et les points de repère » à la page 549.

Utilisation des métadonnées de la vidéo

Vous pouvez utiliser les fonctions `OnMetaData()` et `OnXMPData()` pour accéder à des informations sur les métadonnées dans votre fichier vidéo, y compris les points de repère.

Utilisation d'OnMetaData()

Ces métadonnées comprennent des informations sur le fichier vidéo (durée, largeur et hauteur d'image, cadence). Les informations de métadonnées ajoutées à votre fichier vidéo dépendent du logiciel que vous utilisez pour coder le fichier vidéo.

```
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]);
    }
}
```

Le code précédent génère un résultat similaire à celui-ci :

```
width: 320
audiodelay: 0.038
canSeekToEnd: true
height: 213
cuePoints: ,,
audiodatarate: 96
duration: 16.334
videodatarate: 400
framerate: 15
videocodecid: 4
audiocodecid: 2
```

Si la vidéo ne contient pas de son, les informations de métadonnées associées à l'audio (telles que `audiodatarate`) renvoient `undefined`, car aucune information audio n'est ajoutée aux métadonnées pendant l'encodage.

Dans le code ci-dessus, les informations de point de repère n'étaient pas affichées. Pour afficher les informations de point de repère, utilisez la fonction suivante, qui affiche de façon récursive les éléments d'un objet :

```
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);
        }
    }
}
```

L'utilisation du code ci-dessus pour suivre le paramètre `infoObject` de la méthode `onMetaData()` produit le résultat suivant :

```
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
```

L'exemple suivant affiche les métadonnées d'une vidéo MP4. Cet exemple suppose qu'il existe un objet TextArea appelé `metaDataOut` sur lequel il écrit les métadonnées.

```
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;
        }
    }
}
```

La fonction `onMetaData()` a produit le résultat suivant pour cette vidéo :

```
moovposition=731965
height=352
avclevel=21
videocodecid=avc1
duration=2.36
width=704
videoframerate=25
avcprofile=88
trackinfo=[object Object]
```

Utilisation de l'objet Information

Le tableau ci-dessous indique les valeurs possibles pour des métadonnées de vidéo qui sont transmises à la fonction de rappel `onMetaData()` dans l'Objet qu'elles reçoivent :

Paramètre	Description
aacaot	Type d'objet audio AAC ; 0, 1 ou 2 pris en charge.
avclevel	Numéros de niveau AVC IDC, tels que 10, 11, 20, 21, et ainsi de suite.
avcprofile	Numéros de profil AVC, tel que 55, 77, 100, et ainsi de suite.
audiocodecid	Chaîne qui indique le codec audio (technique de codage/décodage) utilisé ; par exemple, « .Mp3 » ou « mp4a ».
audiodatarate	Nombre qui indique le taux d'encodage du son, en ko/s.
audiodelay	Nombre qui indique la valeur temporelle 0 du fichier FLV d'origine. Le contenu vidéo doit être légèrement retardé pour synchroniser correctement l'audio.

Paramètre	Description
canSeekToEnd	Valeur booléenne définie sur `true` si le fichier FLV est codé avec une image-clé sur la dernière image, qui permet de rechercher jusqu'à la fin d'un fichier vidéo téléchargé progressivement. Elle est définie sur `false` si le fichier FLV n'est pas codé avec une image-clé sur la dernière image.
cuePoints	Tableau d'objets (un par point de repère intégré dans le fichier FLV). Cette valeur n'est pas définie si le fichier FLV ne contient pas de points de repère. Chaque objet possède les propriétés ci-dessous. • `type` : chaîne qui spécifie le type de point de repère : « navigation » ou « event ». • `name` : chaîne représentant le nom du point de repère. • `time` : nombre correspondant à l'heure du point de repère (en secondes) avec une précision de trois chiffres (millisecondes). • `parameters` : objet facultatif possédant des paires nom-valeur désignées par l'utilisateur au moment de la création des points de repère.
duration	Nombre indiquant la durée du fichier vidéo, en secondes.
framerate	Nombre indiquant la fréquence d'images du fichier FLV.
height	Nombre indiquant la hauteur du fichier FLV, en pixels.
seekpoints	Tableau qui répertorie les images-clés disponibles en tant que cachets d'horodatage, en millisecondes. Facultatif.
balises	Tableau de paires clé-valeur qui représente les informations dans l'atome « ilst » (l'équivalent des balises ID3 des fichiers MP4). iTunes utilise ces balises. Elles peuvent être utilisées pour afficher des illustrations, le cas échéant.
trackinfo	Objet qui fournit des informations sur toutes les pistes d'un fichier MP4, y compris sur l'ID de description de l'échantillonnage.
videocodecid	Chaîne indiquant la version codec utilisée pour coder la vidéo. Par exemple : « avc1 » ou « VP6F ».
videodatarate	Nombre indiquant la vitesse de transmission vidéo du fichier FLV.
videoframerate	Cadence de la vidéo MP4.
width	Nombre indiquant la largeur du fichier FLV, en pixels.

Le tableau suivant répertorie les valeurs possibles du paramètre `videocodecid` :

videocodecid	Nom du codec
2	Sorenson H.263
3	Screen video (SWF versions 7 et ultérieures uniquement)
4	VP6 (SWF versions 8 et ultérieures uniquement)
5	VP6 avec canal alpha (SWF versions 8 et ultérieures uniquement)

Le tableau suivant répertorie les valeurs possibles du paramètre `audiocodecid` :

audiocodecid	Nom du codec
0	non compressé
1	ADPCM
2	Mp3
4	Nellymoser @ 16 kHz mono

audiocodecid	Nom du codec
5	Nellymoser, 8kHz mono
6	Nellymoser
10	AAC
11	Speex

Utilisation d'onXMPData()

La fonction de rappel `onXMPData()` reçoit des informations spécifiques à Adobe Extensible Metadata Platform (XMP) intégrée dans le fichier vidéo Adobe F4V ou FLV. Les métadonnées XMP contiennent des points de repère et d'autres métadonnées de vidéo. Les métadonnées XMP sont présentées dans Flash Player 10 et Adobe AIR 1.5, et elles sont prises en charge par les versions ultérieures de Flash Player et d'Adobe AIR.

L'exemple suivant traite les données de points de repère dans des métadonnées XMP :

```
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);
        }
    }
}
```

Pour un fichier vidéo court appelé startrekintro.f4v, cet exemple produit les lignes de suivi ci-dessous. Les lignes montrent les données des points de repère pour les points de repère de navigation et d'événement dans les métadonnées XMP :

```
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
```

Remarque : *dans les données XMP, le temps est stocké en unités DVA plutôt qu'en secondes Pour calculer le temps du point de repère, divisez l'heure de démarrage par la cadence. Par exemple, l'heure de démarrage de 7695905817600 divisé par une cadence de 254016000000 est égal à 30:30.*

Pour voir les métadonnées XMP brutes au complet, qui contiennent la cadence, retirez les identificateurs de commentaires (//) qui précèdent les deuxièeme et troisième instructions `trace()` au début de la fonction `onXMPData()`.

Pour plus d'informations sur XMP, consultez :

- http://partners.adobe.com/public/developer/xmp/topic.html
- http://www.adobe.com/devnet/xmp/

Utilisation des métadonnées de l'image

L'événement `onImageData` envoie les données d'image sous la forme d'un tableau d'octets par l'intermédiaire d'un canal de données AMF0. Les données peuvent être au format JPEG, PNG ou GIF. Définissez une méthode de rappel `onImageData()` pour traiter ces informations, de la même manière que vous définiriez des méthodes de rappel pour `onCuePoint` et `onMetaData`. L'exemple suivant accède aux données d'image et les affiche à l'aide d'une méthode de rappel `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);
}
```

Utilisation des métadonnées du texte

L'événement `onTextData` envoie des données de texte par le biais d'un canal de données AMF0. Les données de texte sont au format UTF-8 et contiennent des informations supplémentaires sur la mise en forme basées sur la spécification Timed Text 3GP. Cette spécification définit un format de sous-titrage normalisé. Définissez une méthode de rappel `onTextData()` pour traiter ces informations, de la même manière que vous définiriez des méthodes de rappel pour `onCuePoint` ou `onMetaData`. Dans l'exemple suivant, la méthode `onTextData()` affiche le numéro d'identification de la piste, ainsi que le texte correspondant à la piste.

```
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);
}
```

Capture d'un signal vidéo provenant de la caméra de l'utilisateur

Outre un fichier vidéo externe, la source des données vidéo peut également être une caméra connectée à l'ordinateur de l'utilisateur. Il est possible de gérer en ActionScript l'affichage et la manipulation de ces données. La classe Camera est le mécanisme intégré à ActionScript pour la gestion d'une caméra reliée à l'ordinateur.

Présentation de la classe Camera

L'objet Camera permet d'établir une connexion avec la caméra locale de l'utilisateur et de diffuser le signal vidéo soit localement (à destination de l'utilisateur lui-même) ou à destination d'un serveur tel que Flash Media Server.

La classe Camera permet d'accéder aux informations suivantes sur la caméra de l'utilisateur :

- Les caméras qui sont installées sur l'ordinateur et disponibles pour Flash Player
- Si une caméra est installée

- Si Flash Player est autorisé ou non à accéder à la caméra de l'utilisateur
- Quelle est la caméra active
- La largeur et la hauteur de la vidéo en cours de capture, en pixels.

La classe Camera fournit les méthodes et les propriétés qui permettent d'utiliser les objets Camera. Par exemple, la propriété statique `Camera.names` contient le tableau des noms des caméras actuellement installées sur l'ordinateur. Par ailleurs, la propriété `name` permet d'afficher le nom de la caméra active.

Affichage du contenu de la caméra

La connexion à une caméra peut nécessiter moins de code que l'utilisation des classes NetConnection et NetStream pour charger un fichier vidéo. Par contre, l'utilisation de la classe Camera peut parfois s'avérer délicate, car avec Flash Player, il est nécessaire d'avoir l'autorisation de l'utilisateur pour se connecter à sa caméra et la rendre accessible par code.

Le code suivant montre comment utiliser la classe Camera pour établir une connexion avec la caméra de l'utilisateur:

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Remarque : *la classe Camera ne possède pas de méthode constructeur. Pour créer une nouvelle occurrence de Camera, utilisez la méthode statique* `Camera.getCamera()`.

Conception d'une application gérant une caméra locale

Lors de la programmation d'une application destinée à gérer la caméra de l'utilisateur, prenez les précautions suivantes :

- Vérifiez qu'une caméra est installée sur l'ordinateur de l'utilisateur.
- Pour Flash Player uniquement, vérifiez si l'utilisateur a autorisé l'accès à la caméra de façon explicite. Pour des raisons de sécurité, le lecteur Flash affiche la boîte de dialogue de paramétrage de Flash Player, qui permet à l'utilisateur d'autoriser ou non l'accès à sa caméra. Flash Player ne peut donc pas établir une connexion avec la caméra d'un utilisateur et diffuser un signal vidéo sans la permission de cet utilisateur. Si l'utilisateur clique sur le bouton d'autorisation, votre application peut se connecter à sa caméra. Si l'utilisateur clique sur le bouton de refus, votre application ne pourra pas établir de liaison avec sa caméra. Dans les deux cas, votre application doit gérer élégamment la réponse.

Etablissement d'une connexion avec la caméra de l'utilisateur

Pour établir une connexion avec la caméra de l'utilisateur, la première étape consiste à créer une occurrence de l'objet Camera, en créant une variable du type Camera et en l'initialisant avec la valeur renvoyée par la méthode statique `Camera.getCamera()`.

L'étape suivante consiste à créer un objet Video et à lui affecter l'objet Camera.

La troisième étape consiste à ajouter cet objet Video à la liste d'affichage. Les étapes 2 et 3 sont nécessaires, car la classe Camera n'étend pas la classe DisplayObject, il est donc impossible de l'ajouter directement à la liste d'affichage. Pour afficher le signal vidéo provenant de la caméra, vous devez ensuite créer un autre objet Video et appeler la méthode `attachCamera()`.

Le code suivant illustre ces trois opérations :

```
var cam:Camera = Camera.getCamera();
var vid:Video = new Video();
vid.attachCamera(cam);
addChild(vid);
```

Notez que si aucune caméra n'est installée sur l'ordinateur de l'utilisateur, l'application n'affiche rien.

Pour une application réelle, d'autres tâches sont nécessaires. Pour plus d'informations, consultez les sections « Vérification de la présence de caméras » à la page 565 et « Détection de l'autorisation d'accéder à la caméra » à la page 566.

Vérification de la présence de caméras

Avant de tenter d'utiliser les méthodes ou propriétés d'une occurrence de Camera, il est préférable de vérifier qu'une caméra est bien installée. Il existe deux techniques pour vérifier la présence d'une ou plusieurs caméras :

- Tester la propriété statique `Camera.names`, qui contient le tableau des noms des caméras disponibles. Ce tableau ne comporte en général qu'une seule chaîne au maximum, car la plupart des utilisateurs ne disposent au mieux que d'une seule caméra à la fois. Le code suivant montre comment tester la propriété `Camera.names` pour vérifier que l'utilisateur dispose d'au moins une caméra :

  ```
  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.");
  }
  ```

- Vérifiez la valeur renvoyée de la méthode statique `Camera.getCamera()`. Si aucune caméra n'est disponible ou installée, la méthode renvoie `null`, sinon elle renvoie une référence à un objet Camera. Le code suivant montre comment appeler la méthode `Camera.getCamera()` pour vérifier que l'utilisateur dispose d'au moins une caméra :

  ```
  var cam:Camera = Camera.getCamera();
  if (cam == null)
  {
      trace("User has no cameras installed.");
  }
  else
  {
      trace("User has at least 1 camera installed.");
  }
  ```

Etant donné que la classe Camera n'étend pas la classe DisplayObject, il est impossible de l'ajouter directement à la liste d'affichage à l'aide de la méthode `addChild()`. Pour afficher le signal vidéo provenant de la caméra, vous devez donc créer un objet Video et appeler la méthode `attachCamera()` de l'occurrence de Video.

Ce fragment de code montre comment établir la connexion avec la caméra, s'il en existe une. Dans le cas contraire, Flash Player n'affiche rien :

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video();
    vid.attachCamera(cam);
    addChild(vid);
}
```

Détection de l'autorisation d'accéder à la caméra

Dans le sandbox de l'application AIR, l'application peut accéder à n'importe quelle caméra sans autorisation de l'utilisateur.

Avant que Flash Player ne puisse afficher les résultats d'une caméra, l'utilisateur doit autoriser Flash Player à accéder à la caméra de façon explicite. Lorsque la méthode `attachCamera()` est appelée, la boîte de dialogue Paramètres de Flash Player est affichée pour permettre à l'utilisateur d'autoriser ou refuser à Flash Player l'accès à la caméra et au microphone. Si l'utilisateur a accordé son autorisation, Flash Player affiche les résultats de la caméra dans l'occurrence de l'objet Video sur la scène. Si l'utilisateur a refusé, Flash Player ne peut pas se connecter à la caméra et l'objet Video n'affiche rien.

Pour déterminer si l'utilisateur a accordé à Flash Player l'accès à la caméra, vous pouvez écouter l'événement `status` de la caméra (`StatusEvent.STATUS`), comme dans le code suivant :

```
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"
}
```

La fonction `statusHandler()` est appelée dès que l'utilisateur clique sur Autoriser ou Refuser. Deux méthodes permettent de détecter le bouton qui a été cliqué :

- Le paramètre `event` de la fonction `statusHandler()` possède une propriété code qui contient la chaîne « Camera.Muted » ou « Camera.Unmuted ». Si la valeur est « Camera.Muted », l'utilisateur a refusé l'autorisation et Flash Player ne peut pas accéder à la caméra. Le fragment de code suivant en est un exemple :

  ```
  function statusHandler(event:StatusEvent):void
  {
      switch (event.code)
      {
          case "Camera.Muted":
              trace("User clicked Deny.");
              break;
          case "Camera.Unmuted":
              trace("User clicked Accept.");
              break;
      }
  }
  ```

- La classe Camera contient une propriété en lecture seule appelée `muted` qui indique si l'utilisateur a refusé l'accès à la caméra (`true`) ou l'a autorisé (`false`) dans le panneau Confidentialité de Flash Player. Le fragment de code suivant en est un exemple :

```
function statusHandler(event:StatusEvent):void
{
    if (cam.muted)
    {
        trace("User clicked Deny.");
    }
    else
    {
        trace("User clicked Accept.");
    }
}
```

En consultant l'événement d'état à distribuer, vous pouvez rédiger du code pour gérer l'autorisation ou le refus d'accès à la caméra et terminer proprement. Par exemple, si l'utilisateur a refusé, vous pouvez afficher un message lui expliquant qu'il doit donner son autorisation s'il veut participer à une conversation vidéo, ou au contraire veiller à supprimer l'objet Video de la liste d'affichage afin de libérer des ressources système.

Optimisation de la qualité d'image vidéo

Par défaut, les nouvelles occurrences de la classe Video ont une largeur de 320 pixels sur une hauteur de 240 pixels. Pour optimiser la qualité vidéo, veillez à donner à vos objets Video les mêmes dimensions que celles de l'objet Camera. Pour obtenir la largeur et la hauteur de l'objet Camera, utilisez les propriétés `width` et `height` de la classe Camera. Vous pouvez alors adapter les propriétés `width` et `height` de l'objet Video à ces dimensions, ou transmettre celles-ci à la méthode constructeur de la classe Video, comme dans le fragment de code ci-dessous :

```
var cam:Camera = Camera.getCamera();
if (cam != null)
{
    var vid:Video = new Video(cam.width, cam.height);
    vid.attachCamera(cam);
    addChild(vid);
}
```

Comme la méthode `getCamera()` renvoie une référence à un objet Camera (ou `null` si aucune caméra n'est présente), vous pouvez utiliser les méthodes et les propriétés de cet objet même si l'utilisateur refuse l'accès à la caméra. Vous pouvez ainsi définir les dimensions de l'occurrence de l'objet Video en fonction de celles de la caméra.

```
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);
}
```

Pour plus d'informations sur le mode plein écran, consultez la section relative au mode plein écran sous « Définition des propriétés de la scène » à la page 291.

Suivi des conditions de lecture

La classe Camera contient plusieurs propriétés qui permettent de suivre l'état de l'objet Camera. Par exemple, le code ci-dessous affiche plusieurs propriétés de la caméra à l'aide d'un objet Timer et d'une occurrence de champ de texte dans la liste d'affichage:

```
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");
}
```

Tous les 1/10 de seconde (100 millisecondes) l'événement `timer` de l'objet Timer est diffusé et la fonction `timerHandler()` actualise le contenu du champ de texte dans la liste d'affichage.

Envoi de vidéo à un serveur

Si vous souhaitez créer des applications plus complexes avec des objets Video ou Camera, Flash Media Server (FMS) offre diverses possibilités de diffusion de flux multimédia ainsi qu'un environnement de développement pour créer des applications multimédia et les distribuer à l'intention d'un public très large. Cette combinaison permet aux développeurs de créer des applications telles que vidéo à la demande, diffusion d'événements en direct sur le Web et diffusion en flux continu MP3, mais aussi blog vidéo, vidéomessagerie ou conversation multimédia. Pour plus d'informations, consultez la documentation de Flash Media Server disponible en ligne à l'adresse suivante : www.adobe.com/go/learn_fms_docs_fr.

Rubriques avancées pour les fichiers FLV

Les rubriques suivantes abordent certains problèmes d'utilisation des fichiers FLV.

A propos de la configuration de fichier FLV pour l'hébergement sur un serveur

Pour utiliser des fichiers FLV, il peut être nécessaire de configurer votre serveur pour le format de fichier FLV. Le protocole MIME (Multipurpose Internet Mail Extensions) est une spécification de données normalisée qui permet d'envoyer des fichiers non ASCII via des connexions Internet. Les navigateurs Web et les clients de courrier électronique sont configurés pour interpréter de nombreux types MIME afin qu'ils puissent envoyer et recevoir de la vidéo, de l'audio, des graphiques et du texte formaté. Pour charger des fichiers FLV à partir d'un serveur Web, il peut être nécessaire d'enregistrer l'extension de fichier et le type MIME auprès de votre serveur Web. Pour plus d'informations, consultez la documentation du serveur. Le type MIME des fichiers FLV est `video/x-flv`. Les informations complètes du type de fichier FLV se présentent comme suit :

- Type Mime : video/x-flv
- Extension du fichier : .flv
- Paramètres requis : aucun
- Paramètres facultatifs : aucun
- Considérations sur l'encodage : les fichiers FLV sont binaires, et une partie des applications peut nécessiter la définition du sous-type application/octet-stream
- Problèmes de sécurité : aucun
- Spécifications publiées : www.adobe.com/go/video_file_format_fr

Microsoft a changé la façon dont le multimédia en flux continu est géré par le serveur Web 6.0 IIS (Microsoft Internet Information Services) par rapport à ses versions antérieures. Les versions antérieures d'IIS ne nécessitent aucune modification pour diffuser en continu de la vidéo Flash. Dans IIS 6.0, le serveur Web fourni par défaut avec Windows 2003, le serveur nécessite un type MIME pour reconnaître que les fichiers FLV sont des flux continus multimédia.

Lorsque des fichiers SWF qui diffusent des fichiers FLV externes sont placés sur un serveur Microsoft Windows Server® 2003 et sont affichés dans un navigateur, le fichier SWF est lu correctement, mais la vidéo FLV n'est pas diffusée en continu. Ce problème a une incidence sur tous les fichiers FLV placés sur le serveur Windows Server 2003, y compris les fichiers créés avec des versions antérieures de l'outil de programmation Flash et le Kit Macromedia Flash Video Kit pour Dreamweaver MX 2004 d'Adobe. Ces fichiers fonctionnent correctement si vous les testez avec d'autres systèmes d'exploitation.

Pour plus d'informations sur la méthode de configuration de Microsoft Windows 2003 et Microsoft IIS Server 6.0 pour diffuser en continu de la vidéo FLV, consultez la page www.adobe.com/go/tn_19439_fr.

Ciblage des fichiers FLV locaux sur Macintosh

Si vous tentez de lire un fichier FLV local à partir d'un lecteur non système sur un ordinateur Apple® Macintosh® avec un chemin qui utilise une barre oblique (/), il ne sera pas lu. Les lecteurs non système sont par exemple les lecteurs de CD-ROM, les disques durs partitionnés, les supports de stockage amovibles, les périphériques de stockage connectés, etc.

Remarque : *la raison de cet échec est une limitation du système d'exploitation, et non pas de Flash Player ni d'AIR.*

Pour qu'un fichier FLV puisse être lu à partir d'un lecteur non système sur un Macintosh, faites-y référence à l'aide d'un chemin absolu utilisant une notation à deux points (:) au lieu d'une notation à barres obliques (/). La liste suivante montre la différence entre les deux sortes de notation :

- Notation à barres obliques **:** myDrive/myFolder/myFLV.flv
- Notation à deux points **:** (Mac OS®) myDrive:myFolder:myFLV.flv

Vous pouvez aussi créer un fichier de projection pour un CD-ROM que vous voulez utiliser pour la lecture Macintosh. Pour obtenir les dernières informations sur les CD-ROM Macintosh et les fichiers FLV, consultez la page www.adobe.com/go/3121b301_fr.

Exemple : Video Jukebox

L'exemple suivant crée un jukebox vidéo simple qui charge dynamiquement une liste de fichiers vidéos à lire en séquence. Vous pouvez ainsi créer une application qui permet à l'utilisateur de parcourir une série de didacticiels, ou encore qui permet de définir des publicités à afficher avant la vidéo demandée par l'utilisateur. Cet exemple illustre les fonctions suivantes d'ActionScript 3.0 :

- Actualisation de la position de la tête de lecture en fonction de la progression dans le fichier vidéo
- Détection et analyse des métadonnées d'un fichier vidéo
- Gestion de codes spécifiques dans un flux Internet
- Chargement, lecture, mise en pause et arrêt d'un fichier FLV chargé dynamiquement
- Redimensionnement d'un objet vidéo dans la liste d'affichage en fonction des métadonnées du flux reçu

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers de l'application Video Jukebox se trouvent dans le dossier Samples/VideoJukebox. L'application se compose des fichiers suivants :

Fichier	Description
VideoJukebox.fla ou VideoJukebox.mxml	Le fichier d'application principal pour Flex (MXML) ou Flash (FLA).
VideoJukebox.as	Classe comportant les principales fonctionnalités de l'application.
playlist.xml	Fichier comportant la liste des fichiers vidéo à charger dans le jukebox.

Chargement l'une liste de lecture vidéo externe

Le fichier externe playlist.xml contient la liste des vidéos à charger et leur ordre de lecture. Pour charger ce fichier XML, utilisez un objet URLLoader et un objet URLRequest, comme dans le code ci-dessous :

```
uldr = new URLLoader();
uldr.addEventListener(Event.COMPLETE, xmlCompleteHandler);
uldr.load(new URLRequest(PLAYLIST_XML_URL));
```

Ce code est placé dans le constructeur de la classe VideoJukebox, si bien que le fichier est chargé avant l'exécution de la suite du code. Dès que le chargement du fichier XML est terminé, la méthode `xmlCompleteHandler()` est appelée et analyse le fichier externe dans un objet XML, comme dans le code suivant :

```
private function xmlCompleteHandler(event:Event):void
{
    playlist = XML(event.target.data);
    videosXML = playlist.video;
    main();
}
```

L'objet XML `playlist` contient les données XML brutes du fichier externe, alors que videosXML est un objet XMLList qui ne contient que les nœuds vidéo. Le fragment de code suivant est un exemple de contenu du fichier playlist.xml :

```
<videos>
    <video url="video/caption_video.flv" />
    <video url="video/cuepoints.flv" />
    <video url="video/water.flv" />
</videos>
```

Enfin, la méthode `xmlCompleteHandler()` appelle la méthode `main()` qui définit les diverses occurrences de composants dans la liste d'affichage, ainsi que les objets NetConnection et NetStream qui permettent de charger les fichiers FLV externes.

Création de l'interface utilisateur

Pour créer l'interface utilisateur, faites glisser cinq occurrences de l'objet Button sur la liste d'affichage et donnez-leur les noms d'occurrence suivants : `playButton`, `pauseButton`, `stopButton`, `backButton` et `forwardButton`.

Pour chacune de ces occurrences de Button, affectez un gestionnaire pour l'événement `click`, comme dans le fragment de code suivant :

```
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);
```

La méthode `buttonClickHandler()` utilise une instruction switch pour déterminer l'occurrence de bouton sur laquelle l'utilisateur a cliqué, comme dans le code suivant :

```
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;
    }
}
```

Ajoutez ensuite une occurrence de Slider à la liste d'affichage, et donnez à cette occurrence le nom `volumeSlider`. Le code ci-dessous définit la propriété `liveDragging` de l'occurrence de Slider sur `true` et définit un écouteur d'événement pour l'événement `change` de cette occurrence :

```
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);
```

Ajoutez ensuite une occurrence de ProgressBar à la liste d'affichage, et donnez à cette occurrence le nom `positionBar`. Donnez à sa propriété `mode` la valeur « manual », comme ci-dessous :

```
positionBar.mode = ProgressBarMode.MANUAL;
```

Enfin, ajoutez une occurrence de l'objet Label à la liste d'affichage, et donnez à cette occurrence le nom `positionLabel`. La valeur de cette occurrence de l'objet Label sera définie par l'occurrence de timer.

Détection des métadonnées d'un objet vidéo

Lorsque Flash Player détecte des métadonnées dans l'une des vidéos chargées, le gestionnaire de rappel `onMetaData()` est appelé pour la propriété `client` de l'objet NetStream. Le code suivant initialise un Object et configure le gestionnaire de rappel spécifié :

```
client = new Object();
client.onMetaData = metadataHandler;
```

La méthode `metadataHandler()` copie ses données dans la propriété meta définie par code au préalable. Vous pouvez ainsi accéder à tout moment aux métadonnées de la vidéo active, depuis n'importe quel point de l'application. L'objet Video sur la scène est ensuite redimensionné selon la taille renvoyées par les métadonnées. Enfin, l'occurrence de positionBar de la barre de progression est déplacée et redimensionnée en fonction de la taille de la vidéo en cours. Le code suivant est celui de la méthode `metadataHandler()` :

```
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;
}
```

Chargement dynamique d'une vidéo Flash

Pour charger dynamiquement chacune des vidéos Flash, l'application utilise des objets NetConnection et NetStream. Le code suivant crée un objet NetConnection et passe la valeur `null` à la méthode `connect()`. Cette valeur `null` signifie que Flash Player va se connecter à une vidéo sur l'ordinateur local plutôt que sur un serveur tel que Flash Media Server.

Le code suivant crée les occurrences de NetConnection et NetStream, définit un écouteur d'événement pour l'événement `netStatus` et affecte l'objet `client` à la propriété `client`:

```
nc = new NetConnection();
nc.connect(null);

ns = new NetStream(nc);
ns.addEventListener(NetStatusEvent.NET_STATUS, netStatusHandler);
ns.client = client;
```

La méthode `netStatusHandler()` est appelée en cas de changement d'état de la vidéo. C'est le cas lorsque la lecture de la vidéo débute ou s'arrête, lorsque le signal est mis en mémoire tampon ou en cas d'impossibilité de trouver un flux vidéo. Le code suivant répertorie l'événement `netStatusHandler()` :

```
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.
    }
}
```

Le code précédent évalue la propriété code de l'objet info et filtre les codes « NetStream.Play.Start », « NetStream.Play.StreamNotFound » et « NetStream.Play.Stop ». Les autres codes sont ignorés. Si le flux Internet débute, le code lance l'occurrence de Timer qui actualise la tête de lecture. Si le flux Internet est introuvable ou est interrompu, l'occurrence de Timer est arrêtée et l'application tente de lire la vidéo suivante dans la liste de lecture.

A chaque exécution de Timer, l'occurrence de la barre de progression `positionBar` actualise sa position en appelant la méthode `setProgress()` de la classe ProgressBar, et l'occurrence de Label `positionLabel` est actualisée avec le temps écoulé et la durée totale de la vidéo active.

```
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.
    }
}
```

Contrôle du volume de la vidéo

Vous pouvez contrôler le volume audio de la vidéo chargée dynamiquement par le biais de la propriété `soundTransform` de l'objet NetStream. L'application Video jukebox permet de modifier le volume en changeant la valeur de l'occurrence de Slider `volumeSlider`. Le code suivant montre comment changer le volume en affectant la valeur du composant Slider à un objet SoundTransform qui est lui-même affecté à la propriété `soundTransform` de l'objet NetStream:

```
private function volumeChangeHandler(event:SliderEvent):void
{
    volumeTransform.volume = event.value;
    ns.soundTransform = volumeTransform;
}
```

Contrôle de la lecture de la vidéo

Le reste de l'application contrôle la lecture de la vidéo lorsque la fin du flux vidéo est atteinte ou lorsque l'utilisateur change de vidéo.

La méthode suivante obtient l'adresse URL de la vidéo à partir de l'objet XMLList pour l'index sélectionné :

```
private function getVideo():String
{
    return videosXML[idx].@url;
}
```

La méthode `playVideo()` appelle la méthode `play()` de l'objet NetStream pour charger la vidéo sélectionnée :

```
private function playVideo():void
{
    var url:String = getVideo();
    ns.play(url);
}
```

La méthode `playPreviousVideo()` décrémente l'index vidéo sélectionné, appelle la méthode `playVideo()` pour charger le nouveau fichier vidéo et rend la barre de progression visible :

```
private function playPreviousVideo():void
{
    if (idx > 0)
    {
        idx--;
        playVideo();
        positionBar.visible = true;
    }
}
```

La méthode finale, `playNextVideo()`, incrémente l'index vidéo et rappelle la méthode `playVideo()`. Si la vidéo en cours est la dernière de la liste de lecture, la méthode `clear()` est appelée pour l'objet Video et la propriété `visible` de l'occurrence de la barre de progression est mise à `false`:

```
private function playNextVideo():void
{
    if (idx < (videosXML.length() - 1))
    {
        idx++;
        playVideo();
        positionBar.visible = true;
    }
    else
    {
        idx++;
        vid.clear();
        positionBar.visible = false;
    }
}
```

Chapitre 25 : Utilisation du son

ActionScript est conçu pour des applications immersives, interactives, et le son est un élément des applications immersives puissantes souvent ignoré. Vous pouvez ajouter des effets de son à un jeu vidéo, une réaction acoustique à l'interface utilisateur d'une application, ou même créer un programme qui analyse des fichiers mp3 chargés sur Internet, avec du son au coeur de l'application.

Ce chapitre décrit le chargement de fichiers audio externes et l'utilisation de l'audio incorporé dans un fichier SWF. Il explique comment contrôler l'audio, créer des représentations visuelles des informations de son et capturer le son du microphone d'un utilisateur.

Principes de base de l'utilisation du son

Introduction à l'utilisation du son

Les ordinateurs peuvent capturer et coder l'audio numérique (représentation des informations de son de l'ordinateur), le stocker et le récupérer pour le diffuser sur des hauts-parleurs. Il est possible de lire le son à l'aide d'Adobe® Flash® Player ou Adobe® AIR™ et ActionScript.

Lorsque les données audio sont converties au format numérique, elles possèdent différentes caractéristiques (volume du son, son stéréo ou mono). Lorsque vous lisez un son dans ActionScript, vous pouvez régler ces caractéristiques également (augmenter le volume du son ou faire comme s'il provenait d'une certaine direction, par exemple).

Avant de contrôler un son dans ActionScript, les informations de son doivent être chargées dans Flash Player ou AIR. Vous disposez de cinq façons pour charger des données audio dans Flash Player ou AIR afin de les utiliser avec ActionScript. Vous pouvez charger un fichier de son externe comme, par exemple, un fichier MP3 dans le SWF ; vous pouvez incorporer directement les informations de son dans le fichier SWF lors de sa création; vous pouvez obtenir une entrée audio à l'aide du microphone connecté à l'ordinateur d'un utilisateur ; vous pouvez accéder à des données audio diffusées depuis un serveur ; et vous pouvez travailler avec des données audio générées dynamiquement.

Lorsque vous chargez des données audio depuis un fichier de son externe, vous pouvez commencer par lire le début du fichier audio pendant le chargement du reste des données audio.

Même s'il existe différents formats de fichier audio utilisés pour coder l'audio numérique, ActionScript 3.0, Flash Player et AIR prennent en charge les fichiers audio stockés au format mp3. Ils ne peuvent pas charger ni lire directement des fichiers audio de formats différents (WAV ou AIFF, par exemple).

Lorsque vous utilisez du son dans ActionScript, vous utilisez probablement plusieurs classes issues du package flash.media. Utilisez la classe Sound pour accéder aux informations audio en chargeant un fichier audio ou en affectant une fonction à un événement pour échantillonner des données de son, puis en démarrant la lecture. Une fois que vous avez démarré la lecture d'un son, Flash Player et AIR vous permettent d'accéder à un objet SoundChannel. Etant donné qu'un fichier audio chargé est un son parmi d'autres que vous lisez sur l'ordinateur d'un utilisateur, chaque son individuel lu utilise son objet SoundChannel ; c'est la sortie combinée de tous les objets SoundChannel mixés qui est lue sur les haut-parleurs de l'ordinateur. Vous utilisez l'instance SoundChannel pour contrôler les propriétés du son et arrêter sa lecture. Enfin, si vous souhaitez contrôler l'audio combiné, la classe SoundMixer vous permet de contrôler la sortie mixée.

Vous pouvez également utiliser d'autres classes pour effectuer des tâches plus spécifiques lorsque vous utilisez du son dans ActionScript; pour plus d'informations sur toutes les classes liées au son, consultez la section « Présentation de l'architecture audio » à la page 579.

Tâches courantes d'utilisation du son

Ce chapitre décrit les tâches relatives au son suivantes que vous voudrez probablement exécuter :

- Chargement de fichiers mp3 externes et suivi de la progression
- Lecture, pause, reprise et arrêt des sons
- Lecture de sons en continu pendant leur chargement
- Manipulation du volume du son et de la balance
- Récupération de métadonnées ID3 d'un fichier mp3
- Utilisation de données d'onde acoustique brutes
- Génération dynamique du son
- Capture et relecture d'une entrée de son depuis le microphone d'un utilisateur

Concepts importants et terminologie

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Amplitude : distance d'un point sur la courbe audio à partir de la ligne zéro ou d'équilibre.
- Débit : quantité de données codées ou diffusées en continu pour chaque seconde d'un fichier audio. Pour les fichiers mp3, le débit est généralement exprimé en milliers de bits par seconde (kbits/s). Un débit supérieur est souvent synonyme d'une onde acoustique de meilleure qualité.
- Mise en mémoire tampon : la réception et le stockage de données audio avant leur lecture.
- mp3 : MPEG-1 Audio Layer 3, ou mp 3, est un format de compression audio connu.
- Balance horizontale : positionnement d'un signal audio entre les canaux gauche et droit dans un champ acoustique stéréo.
- Crête : point le plus élevé dans une courbe audio.
- Fréquence d'échantillonnage : définit le nombre d'échantillons par seconde extraits d'un signal audio analogique pour créer un signal numérique. La fréquence d'échantillonnage d'un CD audio standard est de 44,1 kHz ou 44 100 échantillons par seconde.
- Lecture en continu : processus consistant à lire les premières portions d'un fichier audio ou d'un fichier vidéo pendant le chargement des dernières portions de ce fichier depuis un serveur.
- Volume : intensité d'un son.
- Courbe audio : forme d'un graphique des différentes amplitudes d'un signal audio au cours du temps.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans le chapitre, vous pouvez tester des exemples de code. Etant donné que ce chapitre traite de l'utilisation du son dans ActionScript, de nombreux exemples impliquent l'utilisation d'un fichier audio (lecture, arrête de la lecture ou réglage du son). Pour tester les exemples de de ce chapitre :

1. Créez un document Flash et enregistrez-le sur votre ordinateur.

2 Dans le scénario, sélectionnez la première image-clé et ouvrez le panneau Actions.

3 Copiez l'exemple de code dans le panneau Script.

4 Si le code implique le chargement d'un fichier audio externe, il aura une ligne de code du type:

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
s.play();
```

où click.mp3 est le nom du fichier audio en cours de chargement. Pour tester ces exemples, vous devez avoir un fichier mp3. Vous devez placer le fichier mp3 dans le même dossier que votre document Flash. Vous devez ensuite modifier le code et utiliser le nom de votre fichier mp3 au lieu du nom de l'exemple de code (par exemple, dans le code ci-dessus, vous remplaceriez click.mp3 par le nom de votre fichier mp3).

5 Dans le menu principal, choisissez Contrôle > Tester l'animation pour créer le fichier SWF et avoir un aperçu du résultat de l'exemple (et l'écouter).

Outre la lecture audio, certains exemples affichent des valeurs à l'aide de la fonction `trace()` ; lorsque vous les testez, les résultats de ces valeurs s'affichent dans le panneau Sortie. Certains exemples dessinent également un contenu à l'écran. Par conséquent, le contenu pour ces exemples s'affiche également dans la fenêtre Flash Player ou AIR.

Pour plus d'informations sur les tests des exemples de code du présent manuel, consultez la section « Test des exemples de code contenus dans un chapitre » à la page 36.

Présentation de l'architecture audio

Vos applications peuvent charger des données audio à partir de cinq sources principales :

- Fichiers audio externes chargés lors de l'exécution
- Ressources audio incorporées dans le fichier SWF de l'application
- Données audio issues d'un microphone connecté au système de l'utilisateur
- Données audio diffusées en continu depuis une passerelle multimédia telle que Flash Media Server
- Données audio générées dynamiquement par le biais du gestionnaire d'événement `sampleData`

Vous pouvez charger entièrement les données audio avant leur lecture, ou bien les lire pendant leur chargement.

ActionScript 3.0 prend en charge les fichiers audio stockés au format mp3. Ils ne peuvent pas charger ni lire directement des fichiers audio de formats différents, tels que WAV ou AIFF. Cependant, à partir de Flash Player 9.0.115.0, les fichiers audio AAC peuvent être chargés et lus à l'aide de la classe NetStream. Il s'agit de la même technique que celle utilisée pour le chargement et la lecture de contenu vidéo. Pour plus d'informations sur cette technique, consultez la section « Utilisation de la vidéo » à la page 536.

Vous pouvez utiliser Adobe Flash CS4 Professional pour importer des fichiers audio WAV ou AIFF puis les intégrer dans les fichiers SWF de votre application au format mp3. L'outil de programmation Flash vous permet également de compresser des fichiers audio intégrés pour réduire leur taille (mais ceci se fait au détriment de la qualité du son). Pour plus d'informations, consultez la section « Importation de sons » dans *Utilisation de Flash*.

L'architecture audio d'ActionScript 3.0 utilise les classes suivantes dans le package flash.media.

Classe	Description
flash.media.Sound	La classe Sound gère le chargement du son, les propriétés de son de base et lance une lecture audio.
flash.media.SoundChannel	Lorsqu'une application lit un objet Sound, un objet SoundChannel est créé pour contrôler la lecture. L'objet SoundChannel contrôle le volume des canaux de lecture gauche et droit du son. Chaque son lu possède son propre objet SoundChannel.
flash.media.SoundLoaderContext	La classe SoundLoaderContext définit le nombre de secondes de mise en mémoire tampon à utiliser lors du chargement d'un son, et indique si Flash Player ou AIR recherche un fichier de régulation sur le serveur lors du chargement d'un fichier. Un objet SoundLoaderContext est utilisé comme paramètre pour la méthode `Sound.load()`.
flash.media.SoundMixer	La classe SoundMixer contrôle les propriétés de lecture et de sécurité de tous les sons dans une application. En effet, plusieurs canaux audio sont mixés au moyen d'un objet SoundMixer commun. Par conséquent, les valeurs de propriété dans l'objet SoundMixer affectent tous les objets SoundChannel en cours de lecture.
flash.media.SoundTransform	La classe SoundTransform contient des valeurs qui contrôlent le volume du son et la balance. Vous pouvez appliquer un objet SoundTransform à un objet SoundChannel individuel, à l'objet SoundMixer global, ou à un objet Microphone, entre autres.
flash.media.ID3Info	Un objet ID3Info contient des propriétés qui représentent les informations de métadonnées ID3 souvent stockées dans des fichiers audio mp3.
flash.media.Microphone	La classe Microphone représente un microphone ou un autre périphérique d'entrée de son connecté à l'ordinateur de l'utilisateur. L'entrée audio issue d'un microphone peut être acheminée vers des haut-parleurs locaux ou envoyée à un serveur distant. L'objet Microphone contrôle le gain, la fréquence d'échantillonnage et d'autres caractéristiques de son propre flux de son.

Chaque son chargé et lu nécessite sa propre occurrence des classes Sound et SoundChannel. La sortie issue de plusieurs occurrences SoundChannel est ensuite mixée par la classe SoundMixer globale pendant la lecture.

Les classes Sound, SoundChannel, et SoundMixer ne sont pas utilisées pour les données audio provenant d'un microphone ou d'une transmission de passerelle multimédia en continu (Flash Media Server, par exemple).

Chargement de fichiers audio externes

Chaque occurrence de la classe Sound permet de charger et de déclencher la lecture d'une ressource audio spécifique. Une application ne peut pas réutiliser un objet Sound pour charger plusieurs sons. Si elle souhaite charger une nouvelle ressource audio, elle doit créer un objet Sound.

Si vous chargez un fichier audio de petite taille (un son clic à associer à un bouton, par exemple), votre application peut créer un objet Sound qui charge automatiquement le fichier audio, comme indiqué ci-dessous :

```
var req:URLRequest = new URLRequest("click.mp3");
var s:Sound = new Sound(req);
```

Le constructeur `Sound()` accepte un objet URLRequest comme premier paramètre. Lorsqu'une valeur pour le paramètre URLRequest est fournie, le nouvel objet Sound commence à charger automatiquement la ressource audio spécifiée.

Dans le meilleur des cas, votre application doit surveiller la progression du chargement du son et rechercher les erreurs pendant le chargement. Par exemple, si le son clic est volumineux, il risque de ne pas être totalement chargé lorsque l'utilisateur clique sur le bouton qui déclenche le son. Si vous tentez de lire un son non chargé, une erreur d'exécution risque de se produire. Il est préférable d'attendre la fin du chargement du son avant de permettre aux utilisateurs d'effectuer des actions risquant de lancer la lecture des sons.

Un objet Sound envoie plusieurs événements différents pendant le chargement du son. Votre application peut écouter ces événements pour suivre la progression du chargement et vérifier que le son est complètement chargé avant la lecture. Le tableau suivant répertorie les événements pouvant être envoyés par un objet Sound.

Evénement	Description
open (`Event.OPEN`)	Envoyé juste avant le début du chargement du son.
progress (`ProgressEvent.PROGRESS`)	Envoyé régulièrement pendant le chargement du son lorsque des données sont reçues du fichier ou du flux.
id3 (`Event.ID3`)	Envoyé lorsque des données ID3 sont disponibles pour un son mp3.
complete (`Event.COMPLETE`)	Envoyé lorsque toutes les données de la ressource audio ont été chargées.
ioError (`IOErrorEvent.IO_ERROR`)	Envoyé lorsqu'un fichier audio est introuvable ou lorsque le chargement est interrompu avant la réception de toutes les données audio.

Le code suivant illustre la lecture d'un son après son chargement:

```
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();
}
```

Tout d'abord, l'exemple de code crée un objet Sound sans lui donner de valeur initiale pour le paramètre URLRequest. Ensuite, il écoute l'événement `Event.COMPLETE` issu de l'objet Sound. La méthode `onSoundLoaded()` s'exécute alors lorsque toutes les données audio sont chargées. Puis, il appelle la méthode `Sound.load()` avec une nouvelle valeur URLRequest pour le fichier audio.

La méthode `onSoundLoaded()` s'exécute lorsque le chargement du son est terminé. La propriété `target` de l'objet Event est une référence à l'objet Sound. L'appel à la méthode `play()` de l'objet Sound lance ensuite la lecture du son.

Surveillance du chargement du son

Les fichiers audio peuvent être très volumineux et leur chargement très long. Flash Player et AIR permettent à votre application de lire des sons avant leur chargement complet. Vous pouvez indiquer à l'utilisateur la quantité de données audio ayant été chargées et la quantité de son déjà lue.

La classe Sound envoie deux événements permettant d'afficher facilement la progression du chargement d'un son : `ProgressEvent.PROGRESS` et `Event.COMPLETE`. L'exemple suivant indique comment utiliser ces événements pour afficher les informations de progression concernant le son en cours de chargement :

```
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);
}
```

Ce code crée d'abord un objet Sound puis lui ajoute des écouteurs pour les événements `ProgressEvent.PROGRESS` et `Event.COMPLETE`. Une fois que la méthode `Sound.load()` a été appelée et que les premières données sont reçues du fichier audio, un événement `ProgressEvent.PROGRESS` a lieu et déclenche la méthode `onSoundLoadProgress()`.

Le pourcentage des données audio chargées est équivalent à la valeur de la propriété `bytesLoaded` de l'objet ProgressEvent divisé par la valeur de la propriété `bytesTotal`. Les mêmes propriétés `bytesLoaded` et `bytesTotal` sont disponibles sur l'objet Sound également. L'exemple ci-dessus indique les messages relatifs à la progression du chargement du son, mais vous pouvez facilement utiliser les valeurs `bytesLoaded` et `bytesTotal` pour mettre à jour les composants de la barre de progression tels que ceux fournis avec la structure d'Adobe Flex 3 ou l'outil de programmation de Flash.

Cet exemple indique également comment une application peut reconnaître et répondre à une erreur lors du chargement des fichiers audio. Par exemple, si un fichier audio avec le nom de fichier donné est introuvable, un événement `Event.IO_ERROR` est envoyé par l'objet Sound. Dans le code précédent, la méthode `onIOError()` s'exécute et affiche un message d'erreur court lorsqu'une erreur se produit.

Utilisation des sons intégrés

Utilisez des sons intégrés (au lieu de charger du son depuis un fichier externe) surtout dans le cas de fichiers audio de petite taille servant d'indicateurs dans l'interface utilisateur de votre application (des sons qui sont lus lorsque vous cliquez sur des boutons, par exemple).

Lorsque vous incorporez un fichier audio dans votre application, la taille du fichier SWF résultant augmente proportionnellement à la taille du fichier audio. Ceci signifie que lorsque vous incorporez des fichiers volumineux dans votre application, la taille de votre fichier SWF risque de devenir trop importante.

La méthode exacte à utiliser pour incorporer un fichier de police dans le fichier SWF de l'application varie selon l'environnement de développement.

Utilisation d'un fichier audio incorporé dans Flash

L'outil de programmation Flash vous permet d'importer des sons dans un grand nombre de formats audio et de les stocker comme symboles dans la bibliothèque. Vous pouvez ensuite les affecter à des images dans le scénario ou aux images d'un état de bouton, les utiliser avec des comportements ou directement dans du code ActionScript. Cette section décrit comment utiliser des sons incorporés dans du code ActionScript avec l'outil de programmation Flash. Pour plus d'informations sur les autres façons d'utiliser des sons incorporés dans Flash, consultez la section « Importation de sons » dans *Utilisation de Flash*.

Pour intégrer un fichier son à l'aide de l'outil de programmation Flash :

1. Sélectionnez Fichier > Importer > Importer dans la bibliothèque, puis sélectionnez un fichier audio et importez-le.
2. Cliquez avec le bouton droit de la souris sur le nom du fichier importé dans le panneau Bibliothèque, et sélectionnez Propriétés. Activez la case à cocher Exporter pour ActionScript.
3. Dans le champ Classe, entrez un nom à utiliser lorsque vous faites référence à ce son incorporé dans ActionScript. Par défaut, il utilisera le nom du fichier audio dans ce champ. Si le nom du fichier contient un point (comme dans DrumSound.mp3), vous devez le remplacer par DrumSound ; ActionScript n'autorise pas le caractère point dans les noms de classe. Le champ Classe de base devrait encore afficher flash.media.Sound.
4. Cliquez sur OK. Il se peut qu'une boîte de dialogue indiquant qu'une définition pour cette classe est introuvable dans le chemin de classe apparaisse. Cliquez sur OK et continuez. Si vous avez saisi un nom qui ne correspond pas à celui d'une classe contenue dans le chemin de classe de votre application, une nouvelle classe qui hérite de la classe flash.media.Sound est générée automatiquement.
5. Pour utiliser le son incorporé, vous référencez le nom de classe pour ce son dans ActionScript. Par exemple, le code suivant commence par créer une occurrence de la classe DrumSound générée automatiquement :

   ```
   var drum:DrumSound = new DrumSound();
   var channel:SoundChannel = drum.play();
   ```

   DrumSound est une sous-classe de la classe flash.media.Sound. Par conséquent, elle hérite des méthodes et des propriétés de la classe Sound, notamment de la méthode `play()` comme indiqué ci-dessus.

Utilisation de fichiers audio de lecture en continu

Lorsqu'un fichier audio ou un fichier vidéo est lu alors que ses données sont encore en cours de chargement, il est *lu en continu*. Les fichiers audio externes chargés depuis un serveur distant sont souvent lus en continu de façon à ce que l'utilisateur ne doive pas attendre le chargement complet des données audio pour écouter le son.

La propriété `SoundMixer.bufferTime` représente le nombre de millisecondes de données audio que Flash Player ou AIR doit rassembler avant la lecture du son. En d'autres termes, si la propriété `bufferTime` est définie sur 5000, Flash Player ou AIR charge au moins 5 000 millisecondes de données depuis le fichier audio avant le début de la lecture du son. La valeur `SoundMixer.bufferTime` par défaut est 1000.

Votre application peut ignorer la valeur `SoundMixer.bufferTime` globale pour un son individuel en spécifiant explicitement une nouvelle valeur `bufferTime` lors du chargement du son. Pour ignorer la durée du tampon par défaut, créez d'abord une occurrence de la classe SoundLoaderContext, définissez sa propriété `bufferTime`, puis transmettez-la comme paramètre à la méthode `Sound.load()`, comme indiqué ci-dessous :

```
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();
```

Pendant la lecture, Flash Player ou AIR tente de conserver le tampon audio à la même taille ou à une taille supérieure. Si le téléchargement des données audio est plus rapide que la vitesse de la lecture, cette dernière continue sans interruption. Néanmoins, si la vitesse de chargement des données est ralentie en raison des limites du réseau, la tête de lecture peut atteindre la fin du tampon audio. Dans ce cas, la lecture est suspendue mais elle reprend automatiquement lorsque d'autres données audio sont chargées.

Pour savoir si la lecture est suspendue car Flash Player ou AIR attend le chargement des données, utilisez la propriété `Sound.isBuffering`.

Utilisation de données audio générées de façon dynamique

Remarque : *Flash Player 10 et Adobe AIR 1.5 donnent désormais la possibilité de générer des données audio de façon dynamique.*

Plutôt que de charger ou de diffuser en continu un son existant, vous pouvez générer des données audio de façon dynamique. Vous pouvez générer des données audio lorsque vous affectez un écouteur associé à l'événement `sampleData` d'un objet Sound. (L'événement `sampleData` est défini dans la classe SampleDataEvent du package flash.events.) Dans cet environnement, l'objet Sound ne charge pas de données audio à partir d'un fichier. Il agit en fait en tant que socket pour les données audio qui lui sont diffusées en continu par l'intermédiaire de la fonction que vous affectez à cet événement.

Lorsque vous ajoutez un écouteur d'événement `sampleData` à un objet Sound, celui-ci demande périodiquement des données à ajouter au tampon audio. Ce tampon contient des données destinées à être lues par l'objet. Une fois appelé, la méthode `play()` de l'objet Sound distribue l'événement `sampleData` lorsqu'il demande de nouvelles données audio. Ceci n'est vrai que si l'objet Sound n'a pas chargé de données mp3 à partir d'un fichier.

L'objet SampleDataEvent comprend une propriété `data`. Dans votre écouteur d'événement, vous écrivez des objets ByteArray dans cet objet `data`. Les tableaux d'octets que vous écrivez dans cet objet s'ajoutent aux données figurant dans le tampon que lit l'objet Sound. Le tableau d'octets que contient le tampon est un flux de valeurs en virgule flottante comprises en -1 et 1. Chaque valeur représente l'amplitude d'un canal unique (gauche ou droit) d'un échantillon audio. Le son est échantillonné à 44 100 échantillons par seconde. Chaque échantillon contient un canal gauche et un canal droit, entrelacés sous forme de données en virgule flottante dans le tableau d'octets.

Dans votre fonction gestionnaire, vous utilisez la méthode `ByteArray.writeFloat()` pour écrire dans la propriété `data` de l'événement `sampleData`. Par exemple, le code suivant génère une onde sinusoïdale :

```
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);
    }
}
```

Lorsque vous appelez `Sound.play()`, l'application commence à appeler votre gestionnaire d'événement pour demander des données audio d'échantillonnage. Elle continue à envoyer des événements pendant la lecture du son jusqu'à ce que vous cessiez de fournir des données ou que vous appeliez la méthode `SoundChannel.stop()`.

La période d'attente de l'événement varie selon les plates-formes et peut encore changer dans les futures versions de Flash Player et AIR. Plutôt que de vous appuyer sur une période d'attente spécifique, calculez-la. Pour calculer la période d'attente, utilisez la formule suivante :

```
(SampleDataEvent.position / 44.1) - SoundChannelObject.position
```

Fournissez entre 2 048 et 8 192 échantillons à la propriété `data` de l'objet SampleDataEvent (pour chaque appel du gestionnaire d'événement). Pour des performances optimales, fournissez autant d'échantillons que possible (8 192 au maximum). Moins vous fournissez d'échantillons, plus il est probable que des bruits parasites se feront entendre pendant la lecture. Ce comportement varie selon les plates-formes et peut se produire dans diverses situations, lors du redimensionnement du navigateur, par exemple. Un code qui fonctionne correctement sur une plate-forme lorsque vous fournissez uniquement 2 048 échantillons ne marchera pas aussi bien sur une autre plate-forme. S'il vous faut le plus court délai d'attente possible, envisagez de permettre à l'utilisateur de sélectionner la quantité de données.

Si vous fournissez moins de 2 048 échantillons (par appel de l'écouteur d'événement `sampleData`), l'application s'arrête à l'issue de la lecture des échantillons restants. Elle distribue ensuite un événement SoundComplete.

Modification du son issu de données MP3

La méthode `SoundExtract` vous permet d'extraire des données d'un objet Sound. Vous pouvez utiliser (et modifier) ces données pour accéder en écriture au flux continu dynamique d'un autre objet Sound à des fins de lecture. Ainsi, le code suivant utilise les octets d'un fichier MP3 chargé et les transmet par le biais d'une fonction de filtre, `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;
}
```

Limitations relatives aux sons générés

Lorsque vous utilisez un écouteur d'événement `sampleData` avec un objet Sound, les seules autres méthodes Sound activées sont `Sound.extract()` et `Sound.play()`. L'appel d'autres méthodes ou propriétés donne lieu à une exception. Tous les méthodes et propriétés de l'objet SoundChannel sont toujours activées.

Lecture de sons

Lire un son chargé peut être aussi simple qu'appeler la méthode `Sound.play()` pour un objet Sound, comme suit :

```
var snd:Sound = new Sound(new URLRequest("smallSound.mp3"));
snd.play();
```

Lorsque vous lisez des sons à l'aide d'ActionScript 3.0, vous pouvez effectuer les opérations suivantes :

- Lire un son à partir d'une position de début spécifique
- Interrompre un son et reprendre la lecture ultérieurement à partir de la même position
- Savoir exactement lorsque la lecture d'un son est terminée
- Suivre la progression de la lecture d'un son
- Modifier le volume ou la balance pendant la lecture d'un son

Pour effectuer ces opérations pendant la lecture, utilisez les classes SoundChannel, SoundMixer et SoundTransform.

La classe SoundChannel contrôle la lecture d'un seul son. La propriété `SoundChannel.position` peut être considérée comme une tête de lecture qui indique le point actuel dans les données audio en cours de lecture.

Lorsqu'une application appelle la méthode `Sound.play()`, une occurrence de la classe SoundChannel est créée pour contrôler la lecture.

Votre application peut lire un son à partir d'une position de début spécifique en la transmettant, en termes de millisecondes, comme paramètre `startTime` de la méthode `Sound.play()`. Elle peut également spécifier un nombre fixe de répétitions du son en succession rapide en transmettant une valeur numérique dans le paramètre `loops` de la méthode `Sound.play()`.

Lorsque la méthode `Sound.play()` est appelée avec un paramètre `startTime` et un paramètre `loops`, le son est lu de façon répétée à partir du même point de début chaque fois, comme indiqué dans le code suivant :

```
var snd:Sound = new Sound(new URLRequest("repeatingSound.mp3"));
snd.play(1000, 3);
```

Dans cet exemple, le son est lu à partir d'un point une seconde après le début du son, trois fois de suite.

Pause et reprise d'un son

Si votre application lit des sons longs (chansons ou podcasts, par exemple), vous pouvez permettre aux utilisateurs d'interrompre et de reprendre leur lecture. Il est impossible d'interrompre littéralement un son pendant la lecture dans ActionScript ; vous pouvez uniquement l'arrêter. Néanmoins, un son peut être lu à partir de n'importe quel point. Vous pouvez enregistrer la position du son au moment de l'arrêt puis le relire ultérieurement à partir de cette position.

Par exemple, supposons que votre code charge et lit un fichier audio de la façon suivante :

```
var snd:Sound = new Sound(new URLRequest("bigSound.mp3"));
var channel:SoundChannel = snd.play();
```

Lors de la lecture du son, la propriété `SoundChannel.position` indique le point dans le fichier audio qui est en cours de lecture. Votre application peut stocker la valeur de position avant d'arrêter la lecture du son, comme suit :

```
var pausePosition:int = channel.position;
channel.stop();
```

Pour reprendre la lecture du son, transmettez la valeur de position stockée précédemment pour relancer le son à partir du même point d'arrêt précédent.

```
channel = snd.play(pausePosition);
```

Surveillance de la lecture

Votre application a peut-être besoin de savoir lorsque la lecture d'un son s'arrête afin de lancer la lecture d'un autre son ou d'effacer des ressources utilisées pendant la lecture précédente. La classe SoundChannel envoie un événement `Event.SOUND_COMPLETE` à la fin de la lecture du son. Votre application peut écouter cet événement et effectuer l'action appropriée, comme indiqué ci-dessous :

```
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.");
}
```

La classe SoundChannel n'envoie pas d'événements progress pendant la lecture. Pour fournir des informations relatives à la progression de la lecture, votre application peut définir son propre mécanisme de synchronisation et suivre la position de la tête de lecture du son.

Pour calculer le pourcentage d'un son lu, vous pouvez diviser la valeur de la propriété `SoundChannel.position` par la longueur des données audio en cours de lecture :

```
var playbackPercent:uint = 100 * (channel.position / snd.length);
```

Néanmoins, ce code signale uniquement des pourcentages de lecture précis si les données audio ont été totalement chargées avant le début de la lecture. La propriété `Sound.length` indique la taille des données audio actuellement chargées, et non pas la taille éventuelle du fichier audio entier. Pour suivre la progression de la lecture d'un son diffusé en continu qui est toujours en cours de chargement, votre application doit estimer la taille éventuelle du fichier audio entier et utiliser cette valeur dans ses calculs. Vous pouvez estimer la longueur éventuelle des données audio à l'aide des propriétés `bytesLoaded` et `bytesTotal` de l'objet Sound, comme suit :

```
var estimatedLength:int =
    Math.ceil(snd.length / (snd.bytesLoaded / snd.bytesTotal));
var playbackPercent:uint = 100 * (channel.position / estimatedLength);
```

Le code suivant charge un fichier audio plus volumineux et utilise l'événement `Event.ENTER_FRAME` comme mécanisme de synchronisation pour afficher la progression de la lecture. Il fournit régulièrement des informations sur le pourcentage de lecture, qui est calculé de la façon suivante : la valeur de position actuelle divisée par la longueur totale des données audio :

```
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);
}
```

Une fois que le chargement des données audio commence, ce code appelle la méthode `snd.play()` et stocke l'objet SoundChannel résultant dans la variable `channel`. Il ajoute ensuite un écouteur d'événement à l'application principale pour l'événement `Event.ENTER_FRAME` et un autre écouteur d'événement à l'objet SoundChannel pour l'événement `Event.SOUND_COMPLETE` qui a lieu à la fin de la lecture.

Chaque fois que l'application atteint une nouvelle image dans son animation, la méthode `onEnterFrame()` est appelée. La méthode `onEnterFrame()` estime la longueur totale du fichier audio en fonction de la quantité de données déjà chargées puis calcule et affiche le pourcentage de lecture actuel.

Une fois que tout le son a été lu, la méthode `onPlaybackComplete()` s'exécute, supprimant l'écouteur d'événement pour l'événement `Event.ENTER_FRAME` de façon à ce qu'il ne tente pas d'afficher les mises à jour de progression après la lecture.

L'événement `Event.ENTER_FRAME` peut être envoyé plusieurs fois par seconde. Dans certains cas, vous pouvez ne pas afficher la progression de la lecture aussi fréquemment. Votre application peut alors définir son propre mécanisme de synchronisation à l'aide de la classe flash.util.Timer ; consultez le chapitre « Utilisation des dates et des heures » à la page 135.

Arrêt de sons diffusés en continu

Il y a quelque chose d'étrange dans le processus de lecture des sons diffusés en continu, c'est-à-dire ceux qui sont lus pendant leur chargement. Lorsque votre application appelle la méthode `SoundChannel.stop()` sur une occurrence de SoundChannel qui lit un son diffusé en continu, la lecture du son s'arrête pendant une image puis elle relance au début du son sur l'image suivante. Ceci a lieu car le chargement du son est toujours en cours. Pour arrêter à la fois le chargement et la lecture d'un son diffusé en continu, appelez la méthode `Sound.close()`.

Sécurité lors du chargement et de la lecture des sons

L'accès aux données audio par votre application peut être limité selon la fonction de sécurité de Flash Player ou AIR. Chaque son est soumis aux restrictions de deux sandbox de sécurité différents, le sandbox pour le contexte lui-même (le sandbox de contexte), et le sandbox pour l'application ou l'objet qui charge et lit le son (le sandbox propriétaire). Pour le contenu de l'application AIR dans le sandbox de sécurité de l'application, tous les sons, y compris ceux chargés à partir d'autres domaines, sont accessibles au contenu dans le sandbox de sécurité de l'application. Toutefois, le contenu dans d'autres sandbox de sécurité observe les même règles que le contenu qui s'exécute dans Flash Player. Pour plus d'informations sur la fonction de sécurité de Flash Player en général et la définition des sandbox, consultez le chapitre « Sécurité dans Flash Player » à la page 714.

Le sandbox de contexte contrôle si des données audio détaillées peuvent être extraites du son à l'aide de la propriété `id3` ou de la méthode `SoundMixer.computeSpectrum()`. Il ne limite pas le chargement ou la lecture du fichier audio lui-même.

Le domaine d'origine du fichier audio définit les limites de sécurité du sandbox de contexte. Généralement, si un fichier audio se trouve dans le même domaine ou dossier que le fichier SWF de l'application ou de l'objet qui le charge, ce dernier dispose d'un accès total à ce fichier audio. Si le son provient d'un domaine différent par rapport à l'application, il peut être intégré dans le sandbox de contexte à l'aide d'un fichier de régulation.

Votre application peut transmettre un objet SoundLoaderContext avec une propriété `checkPolicyFile` comme paramètre à la méthode `Sound.load()`. Lorsque vous définissez la propriété `checkPolicyFile` sur `true`, vous indiquez à Flash Player ou à AIR de rechercher un fichier de régulation sur le serveur à partir duquel le son est chargé. Si un fichier de régulation existe et qu'il autorise l'accès au domaine du fichier SWF de chargement, ce dernier peut charger le fichier audio, accéder à la propriété `id3` de l'objet Sound et appeler la méthode `SoundMixer.computeSpectrum()` pour les sons chargés.

Le sandbox propriétaire contrôle la lecture locale des sons. L'application ou l'objet qui lance la lecture d'un son définit le sandbox propriétaire.

La méthode `SoundMixer.stopAll()` interrompt les sons dans tous les objets SoundChannel en cours de lecture, tant qu'ils répondent aux critères suivants :

- Les sons ont été démarrés par des objets se trouvant dans le même sandbox propriétaire.
- Les sons sont issus d'une source possédant un fichier de régulation qui autorise l'accès au domaine de l'application ou de l'objet qui appelle la méthode `SoundMixer.stopAll()`.

Cependant, dans une application AIR, le contenu du sandbox de sécurité de l'application (contenu installé avec l'application AIR) n'est pas restreint par ces limites de sécurité.

Pour savoir si la méthode `SoundMixer.stopAll()` interrompra tous les sons lus, votre application peut appeler la méthode `SoundMixer.areSoundsInaccessible()`. Si cette méthode renvoie une valeur `true`, certains des sons lus ne sont pas sous le contrôle du sandbox propriétaire actuel et ne seront pas arrêtés par la méthode `SoundMixer.stopAll()`.

La méthode `SoundMixer.stopAll()` empêche également la tête de lecture de continuer pour tous les sons chargés à partir de fichiers externes. Néanmoins, les sons qui sont incorporés dans des fichiers FLA et associés à des images dans le scénario à l'aide de l'outil de programmation Flash risquent d'être relus si l'animation s'est déplacée sur une nouvelle image.

Contrôle du volume du son et de la balance

Un objet SoundChannel individuel contrôle les canaux stéréo gauche et droit pour un son. Si un son mp3 est un son mono, les canaux stéréo gauche et droit de l'objet SoundChannel contiennent des courbes audio identiques.

Vous pouvez connaître l'amplitude de chaque canal stéréo du son lu à l'aide des propriétés `leftPeak` et `rightPeak` de l'objet SoundChannel. Ces propriétés indiquent l'amplitude de crête de la courbe audio du son. Elles ne représentent pas le volume de lecture réel. Le volume de lecture réel est une fonction de l'amplitude de l'onde acoustique et des valeurs de volume définies dans l'objet SoundChannel et la classe SoundMixer.

Vous pouvez utiliser la propriété pan d'un objet SoundChannel pour indiquer un niveau de volume différent pour chacun des canaux gauche et droit pendant la lecture. La propriété pan peut avoir une valeur comprise entre -1 et 1, où -1 signifie que le canal gauche lit à volume maximal alors que le canal droit est muet, et 1 signifie que le canal droit lit à volume maximal alors que le canal gauche est muet. Les valeurs numériques comprises entre -1 et 1 définissent des valeurs proportionnelles pour les valeurs des canaux gauche et droit, et une valeur de 0 signifie que les deux canaux lisent à un niveau de volume moyen, équilibré.

L'exemple de code suivant crée un objet SoundTransform avec une valeur de volume de 0,6 et une valeur de balance horizontale de -1 (volume de canal gauche maximal et aucun volume de canal droit). Il transmet l'objet SoundTransform comme paramètre à la méthode `play()`, qui l'applique au nouvel objet SoundTransform créé pour contrôler la lecture.

```
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);
```

Vous pouvez modifier le volume et la balance pendant la lecture d'un son en définissant les propriétés `pan` ou `volume` d'un objet SoundTransform puis en appliquant cet objet comme propriété `soundTransform` d'un objet SoundChannel.

Vous pouvez également définir des valeurs de balance et de volume global pour tous les sons à la fois à l'aide de la propriété `soundTransform` de la classe SoundMixer, comme l'indique l'exemple suivant :

```
SoundMixer.soundTransform = new SoundTransform(1, -1);
```

Vous pouvez également utiliser un objet SoundTransform pour définir des valeurs de balance et de volume global pour un objet Microphone (consultez la section « Capture de l'entrée de son » à la page 597), et pour des objets Sprite et SimpleButton.

L'exemple suivant modifie la balance horizontale du son du canal gauche au canal droit et de nouveau lors de la lecture du son.

```
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);
}
```

Ce code commence par charger un fichier audio puis crée un objet SoundTransform avec un volume défini sur 1 (volume maximal) et une balance définie sur 0 (balance équilibrée entre gauche et droite). Il appelle ensuite la méthode `snd.play()` en transmettant l'objet SoundTransform comme paramètre.

Lors de la lecture du son, la méthode `onEnterFrame()` s'exécute de façon répétée. La méthode `onEnterFrame()` utilise la fonction `Math.sin()` pour générer une valeur comprise entre -1 et 1 (plage qui correspond aux valeurs acceptables de la propriété `SoundTransform.pan`). La propriété `pan` de l'objet SoundTransform est définie sur la nouvelle valeur, puis la propriété `soundTransform` du canal est définie pour utiliser l'objet SoundTransform modifié.

Pour exécuter cet exemple, remplacez le nom de fichier bigSound.mp3 par le nom d'un fichier mp3 local. Exécutez ensuite l'exemple. Le volume du canal gauche devrait augmenter quand celui du canal droit diminue, et vice-versa.

Dans cet exemple, le même effet peut être obtenu en définissant la propriété `soundTransform` de la classe SoundMixer. Néanmoins, la balance de tous les sons en cours de lecture est affectée (pas seulement le son lu par cet objet SoundChannel).

Utilisation des métadonnées audio

Les fichiers audio qui utilisent le format mp3 peuvent contenir des données supplémentaires relatives au son sous la forme de balises ID3.

Tous les fichiers mp3 ne contiennent pas de métadonnées ID3. Lorsqu'un objet Sound charge un fichier audio mp3, il envoie un événement `Event.ID3` si le fichier audio contient des métadonnées ID3. Pour éviter des erreurs d'exécution, votre application doit attendre de recevoir l'événement `Event.ID3` avant d'accéder à la propriété `Sound.id3` pour un son chargé.

Le code suivant indique comment savoir si les métadonnées ID3 pour un fichier audio ont été chargées :

```
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]);
    }
}
```

Ce code commence par créer un objet Sound et par lui demander d'écouter l'événement `Event.ID3`. Lorsque les métadonnées ID3 du fichier audio sont chargées, la méthode `onID3InfoReceived()` est appelée. La cible de l'objet Event qui est transmise à la méthode `onID3InfoReceived()` est l'objet Sound d'origine. Par conséquent, la méthode obtient ensuite la propriété `id3` de l'objet Sound puis effectue une itération sur toutes ses propriétés appelées pour suivre leurs valeurs.

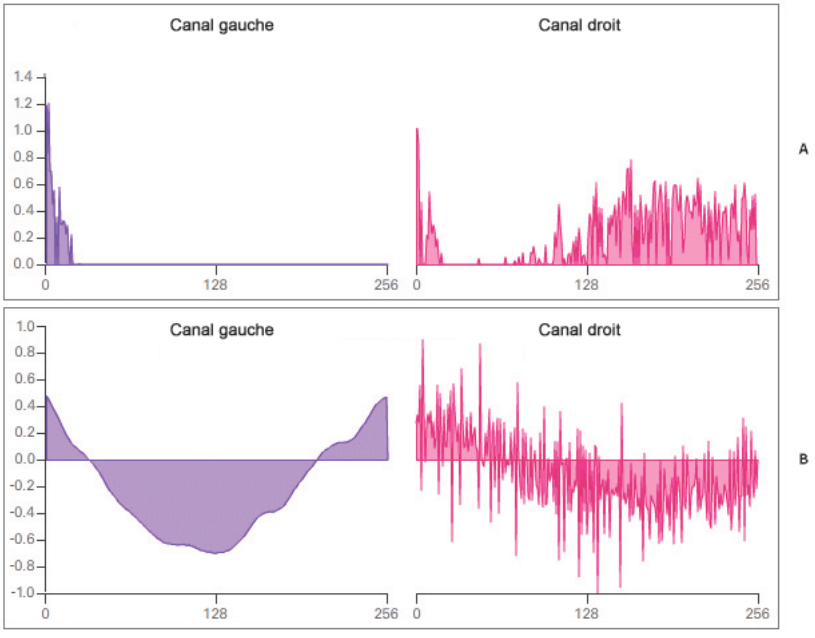
Accès aux données audio brutes

La méthode `SoundMixer.computeSpectrum()` permet à une application de lire les données audio brutes pour la courbe audio en cours de lecture. Si plusieurs objets SoundChannel sont en cours de lecture, la méthode `SoundMixer.computeSpectrum()` indique les données audio combinées de chaque objet SoundChannel mixé.

Les données audio sont renvoyées sous la forme d'un objet ByteArray contenant 512 octets de données (chacun d'eux contenant une valeur en virgule flottante comprise entre -1 et 1). Ces valeurs représentent l'amplitude des points dans la courbe audio en cours de lecture. Les valeurs sont fournies en deux groupes de 256 : le premier groupe pour le canal stéréo gauche et le second pour le canal stéréo droit.

La méthode `SoundMixer.computeSpectrum()` renvoie des données de spectre de fréquences plutôt que des données de courbe audio si le paramètre `FFTMode` est défini sur `true`. Le spectre de fréquences indique l'amplitude par fréquence du son, de la plus basse à la plus élevée. Une FFT (Fast Fourier Transform - transformation de Fourier rapide) est utilisée pour convertir les données de courbe audio en données de spectre de fréquences. Les valeurs de spectre de fréquences résultantes sont comprises entre 0 et 1,414 environ (la racine carrée de 2).

Le diagramme suivant compare les données renvoyées de la méthode `computeSpectrum()` lorsque le paramètre `FFTMode` est défini sur `true` et lorsqu'il est défini sur `false`. Le son dont les données ont été utilisées pour ce diagramme contient un son de basse de grande intensité dans le canal gauche et un son de tambour dans le canal droit.



Valeurs renvoyées par la méthode SoundMixer.computeSpectrum()
A. *fftMode=true* ***B.*** *fftMode=false*

La méthode `computeSpectrum()` peut également renvoyer des données qui ont été rééchantillonnées à un débit inférieur. Généralement, ceci entraîne des données de fréquence ou des données de courbe audio plus lisses, au profit des détails. Le paramètre `stretchFactor` contrôle la fréquence à laquelle les données de la méthode `computeSpectrum()` sont échantillonnées. Lorsque le paramètre `stretchFactor` est défini sur 0 (valeur par défaut), les données audio sont échantillonnées à une fréquence de 44,1 KHz. La fréquence est diminuée de moitié à chaque valeur successive du paramètre stretchFactor. Par conséquent, une valeur de 1 indique une fréquence de 22,05 KHz, une valeur de 2 une fréquence de 11,025 KHz, et ainsi de suite. La méthode `computeSpectrum()` continue à renvoyer 256 octets par canal stéréo lorsqu'une valeur `stretchFactor` supérieure est utilisée.

La méthode `SoundMixer.computeSpectrum()` comporte des limites :

- Etant donné que les données audio issues d'un microphone ou de flux RTMP ne passent pas par l'objet SoundMixer global, la méthode `SoundMixer.computeSpectrum()` ne renvoie pas de données de ces sources.
- Si un ou plusieurs sons lus proviennent de sources externes au sandbox de contexte actuel, les restrictions de sécurité provoquent le renvoi d'une erreur par la méthode `SoundMixer.computeSpectrum()`. Pour plus de détails sur les limites de sécurité de la méthode `SoundMixer.computeSpectrum()`, consultez les sections « Sécurité lors du chargement et de la lecture des sons » à la page 590 et « Accès aux médias chargés comme s'il s'agissait de données » à la page 734.

Cependant, dans une application AIR, le contenu du sandbox de sécurité de l'application (contenu installé avec l'application AIR) n'est pas restreint par ces limites de sécurité.

Création d'un visualiseur audio simple

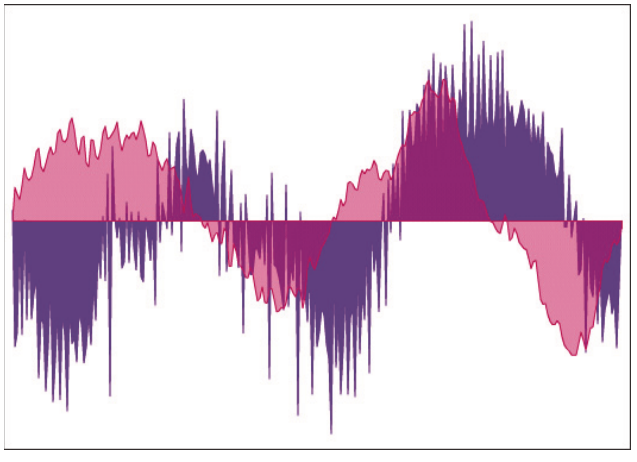
L'exemple suivant utilise la méthode `SoundMixer.computeSpectrum()` pour afficher un diagramme de la courbe audio animée avec chaque image :

```
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);
}
```

Cet exemple commence par charger et lire un fichier audio puis écoute l'événement `Event.ENTER_FRAME` qui déclenchera la méthode `onEnterFrame()` lors de la lecture du son. La méthode `onEnterFrame()` commence par appeler la méthode `SoundMixer.computeSpectrum()`, qui stocke les données d'onde acoustique dans l'objet ByteArray `bytes`.

La courbe audio est tracée à l'aide de l'API de dessin vectoriel. Une boucle `for` passe dans les 256 premières valeurs de données, représentant le canal stéréo gauche, et trace une ligne entre chaque point au moyen de la méthode `Graphics.lineTo()`. Une second boucle `for` passe dans les 256 valeurs suivantes, en les traçant cette fois dans l'ordre inverse, de droite à gauche. Les tracés de courbe audio résultants peuvent produire un effet miroir-image intéressant, comme illustré sur l'image suivante.



Capture de l'entrée de son

La classe Microphone permet à votre application de se connecter à un microphone ou à un autre périphérique d'entrée de son sur le système de l'utilisateur et de diffuser l'audio sur les haut-parleurs de ce système ou d'envoyer les données audio à un serveur distant (Flash Media Server, par exemple). Il est impossible d'accéder à des données audio brutes en provenance du microphone. Vous pouvez uniquement envoyer l'audio aux haut-parleurs du système ou envoyer des données audio compressées à un serveur distant. Pour envoyer des données à un serveur distant, vous pouvez utiliser le codec Speex ou Nellymoser. (Le codec Speex est pris en charge depuis Flash Player version 10 et Adobe AIR version 1.5.)

Accès à un microphone

La classe Microphone ne possède pas de méthode constructeur. A la place, vous utilisez la méthode statique `Microphone.getMicrophone()` pour obtenir une nouvelle occurrence de Microphone, comme indiqué ci-dessous :

```
var mic:Microphone = Microphone.getMicrophone();
```

Lorsque vous appelez la méthode `Microphone.getMicrophone()` sans paramètre, le premier périphérique d'entrée de son détecté sur le système de l'utilisateur est renvoyé.

Un système peut avoir plusieurs périphériques d'entrée de son qui lui sont associés. Votre application peut utiliser la propriété `Microphone.names` pour obtenir un tableau des noms de tous les périphériques d'entrée de son disponibles. Elle peut ensuite appeler la méthode `Microphone.getMicrophone()` avec un paramètre `index` qui correspond à la valeur d'index du nom d'un périphérique dans le tableau.

Il se peut qu'un système n'ait aucun microphone ni périphérique d'entrée de son qui lui soit associé. Vous pouvez utiliser la propriété `Microphone.names` ou la méthode `Microphone.getMicrophone()` pour vérifier si l'utilisateur a installé un périphérique d'entrée de son. Si ce n'est pas le cas, la longueur du tableau `names` est zéro, et la méthode `getMicrophone()` renvoie une valeur `null`.

Lorsque votre application appelle la méthode `Microphone.getMicrophone()`, Flash Player affiche la boîte de dialogue des paramètres de Flash Player, qui invite l'utilisateur à autoriser ou à refuser l'accès Flash Player à la caméra et au microphone sur le système. Une fois que l'utilisateur a fait son choix dans cette boîte de dialogue, un StatusEvent est envoyé. La propriété `code` de cette occurrence de StatusEvent indique si l'accès au microphone a été autorisé ou refusé, comme indiqué dans cet exemple :

```
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.");
    }
}
```

La propriété `StatusEvent.code` contiendra Microphone.Unmuted si l'accès a été autorisé, ou Microphone.Muted s'il a été refusé.

Remarque : *la propriété* `Microphone.muted` *est définie sur* `true` *ou sur* `false` *lorsque l'utilisateur autorise ou refuse l'accès au microphone, respectivement. Néanmoins, la propriété* `muted` *n'est pas définie sur l'occurrence de Microphone jusqu'à l'envoi de StatusEvent. Par conséquent, votre application doit toujours attendre l'envoi de l'événement* `StatusEvent.STATUS` *avant de vérifier la propriété* `Microphone.muted`*.*

Acheminement de l'audio du microphone vers des haut-parleurs locaux

L'entrée audio issue d'un microphone peut être acheminée vers les haut-parleurs du système local en appelant la méthode `Microphone.setLoopback()` avec une valeur de paramètre `true`.

Lorsque le son provenant d'un microphone local est acheminé vers des haut-parleurs locaux, vous risquez de créez une boucle de réaction acoustique pouvant entraîner des grincements d'une grande intensité et endommager ainsi votre matériel. Vous pouvez appeler la méthode `Microphone.setUseEchoSuppression()` avec une valeur de paramètre `true` pour réduire (sans éliminer complètement) le risque de réaction acoustique. Adobe vous conseille de toujours appeler `Microphone.setUseEchoSuppression(true)` avant d'appeler `Microphone.setLoopback(true)`, à moins que vous soyez sûr que l'utilisateur lit le son à l'aide d'un casque ou d'un dispositif autre que les haut-parleurs.

Le code suivant indique comment acheminer l'audio d'un microphone local vers les haut-parleurs du système local :

```
var mic:Microphone = Microphone.getMicrophone();
mic.setUseEchoSuppression(true);
mic.setLoopBack(true);
```

Modification de l'audio du microphone

Votre application peut modifier les données audio provenant d'un microphone de deux façons différentes. Premièrement, elle peut modifier le gain du son entré, qui multiplie les valeurs d'entrée par une quantité spécifiée pour créer un son plus ou moins intense. La propriété `Microphone.gain` accepte des valeurs numériques comprises entre 0 et 100 inclus. Une valeur de 50 a un rôle de multiplicateur de un et spécifie un volume normal. Une valeur de zéro agit comme un multiplicateur de zéro et interrompt l'audio d'entrée. Les valeurs supérieures à 50 indiquent un volume supérieur à la normale.

Votre application peut également modifier la fréquence d'échantillonnage de l'audio d'entrée. Des fréquences d'échantillonnage supérieures augmentent la qualité du son, mais créent également des flux de données plus denses qui utilisent davantage de ressources pour la transmission et le stockage. La propriété `Microphone.rate` représente la fréquence d'échantillonnage audio mesurée en kilohertz (kHz). La fréquence d'échantillonnage par défaut est de 8 kHz. Vous pouvez définir la propriété `Microphone.rate` sur une valeur supérieure à 8 kHz si votre microphone prend en charge la fréquence supérieure. Par exemple, si vous définissez la propriété `Microphone.rate` sur la valeur 11, la fréquence d'échantillonnage est réglée sur 11 kHz ; si vous la définissez sur 22, la fréquence d'échantillonnage est réglée sur 22 kHz, et ainsi de suite. Les fréquences d'échantillonnage disponibles varient en fonction du codec sélectionné. Lorsque vous utilisez le codec Nellymoser, vous pouvez spécifier les fréquences d'échantillonnage 5, 8, 11, 16, 22 et 44 kHz. Lorsque vous utilisez le codec Speex (disponible à partir de Flash Player 10 et Adobe AIR 1.5), vous ne pouvez utiliser que 16 kHz.

Détection de l'activité du microphone

Pour économiser les ressources de traitement et de bande passante, Flash Player tente de détecter lorsque aucun son n'est transmis par un microphone. Lorsque le niveau d'activité du microphone se situe sous le seuil de niveau de silence pendant longtemps, Flash Player arrête la transmission de l'entrée audio et envoie un simple événement ActivityEvent à la place. Si vous utilisez le codec Speex (disponible dans Flash Player 10 et versions ultérieures, ainsi que dans Adobe AIR 1.5 et versions ultérieures), définissez le niveau de silence sur 0 afin de vous assurer que l'application transmet les données audio en continu. La fonction de détection d'activité vocale de Speex réduit automatiquement la bande passante.

Trois propriétés de la classe Microphone surveillent et contrôlent la détection d'activité :

- La propriété `activityLevel` en lecture seule indique la quantité de son détectée par le microphone, sur une échelle de 0 à 100.
- La propriété `silenceLevel` spécifie la quantité de son nécessaire pour activer le microphone et envoie un événement `ActivityEvent.ACTIVITY`. La propriété `silenceLevel` utilise également une échelle de 0 à 100, et la valeur par défaut est 10.
- La propriété `silenceTimeout` décrit le nombre de millisecondes pendant lequel le niveau d'activité doit rester sous le niveau de silence, jusqu'à ce qu'un événement `ActivityEvent.ACTIVITY` soit envoyé pour indiquer que le microphone est maintenant désactivé. La valeur `silenceTimeout` par défaut est 2000.

Les propriétés `Microphone.silenceLevel` et `Microphone.silenceTimeout` sont en lecture seule, mais vous pouvez modifier leurs valeurs à l'aide de la méthode `Microphone.setSilenceLevel()`.

Dans certains cas, l'activation du microphone alors qu'une nouvelle activité est détectée peut entraîner un court délai. Vous pouvez laisser le microphone actif en permanence pour supprimer ces délais d'activation. Votre application peut appeler la méthode `Microphone.setSilenceLevel()` avec le paramètre `silenceLevel` défini sur zéro pour indiquer à Flash Player de laisser le microphone actif et de continuer à rassembler des données audio, même lorsque aucun son n'est détecté. Inversement, lorsque vous définissez le paramètre `silenceLevel` sur 100, le microphone n'est pas activé.

L'exemple suivant affiche les informations relatives au microphone et aux événements activity et status envoyés par un objet Microphone :

```
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);
}
```

Lorsque vous exécutez l'exemple ci-dessus, parlez ou faites du bruit dans votre microphone système et observez les instructions trace qui apparaissent dans une console ou une fenêtre de débogage.

Envoi d'audio vers et depuis une passerelle multimédia

D'autres fonctionnalités audio sont disponibles lorsque vous utilisez ActionScript avec une passerelle multimédia de diffusion en continu telle que Flash Media Server.

Votre application peut notamment associer un objet Microphone à un objet NetStream et transmettre directement des données du microphone de l'utilisateur au serveur. Les données audio peuvent également être diffusées en continu du serveur vers une application avec Flash ou Flex et lues dans le cadre d'un MovieClip ou au moyen d'un objet Video.

Le codec Speex est pris en charge depuis Flash Player version 10 et Adobe AIR version 1.5. Pour définir le codec utilisé pour les données audio compressées envoyées au serveur multimédia, définissez la propriété `codec` de l'objet Microphone. Cette propriété gère deux valeurs, qui sont énumérées dans la classe SoundCodec. La définition de la propriété codec sur `SoundCodec.SPEEX` sélectionne le codec Speex pour la compression audio. La définition de la propriété codec sur `SoundCodec.NELLYMOSER` (valeur par défaut) sélectionne le codec Nellymoser pour la compression audio.

Pour plus d'informations, consultez la documentation en ligne relative à Flash Media Server, à l'adresse suivante : http://livedocs.adobe.com.

Exemple : Podcast Player

Un podcast est un fichier audio distribué sur Internet, sur demande ou sur abonnement. Les podcasts sont généralement publiés dans un annuaire. Etant donné que les épisodes de podcast peuvent durer d'une minute à plusieurs heures, ils sont généralement diffusés en continu pendant la lecture. Les épisodes de podcast, également appelés éléments, sont généralement fournis au format de fichier mp3. Les podcasts vidéo sont également courants, mais cet exemple d'application lit uniquement des podcasts audio utilisant des fichiers mp3.

Cet exemple n'est pas une application agrégatrice de podcasts comprenant toutes les fonctionnalités. Par exemple, elle ne gère pas les abonnements à des podcasts spécifiques et ne mémorise pas les podcasts qu'un utilisateur a écoutés lors de l'exécution suivante de l'application. Il peut servir de point de départ pour un agrégateur de podcasts comprenant toutes les fonctionnalités.

L'exemple Podcast Player illustre les techniques de programmation ActionScript suivantes :

- Lecture d'un fil de syndication et analyse de son contenu XML
- Création d'une classe SoundFacade pour simplifier le chargement et la lecture des fichiers audio
- Affichage de la progression de la lecture du son
- Interruption et reprise de la lecture du son

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application Podcast Player se trouvent dans le dossier Samples/PodcastPlayer. L'application se compose des fichiers suivants :

Fichier	Description
PodcastPlayer.mxml ou PodcastPlayer.fla	Interface utilisateur de l'application pour Flex (MXML) ou Flash (FLA).
comp/example/programmingas3/podcastplayer/PodcastPlayer.as	Classe Document contenant la logique de l'interface utilisateur pour le lecteur de podcast (Flash uniquement).
comp/example/programmingas3/podcastplayer/SoundPlayer.as	Classe pour le symbole du clip SoundPlayer contenant la logique de l'interface utilisateur du lecteur de sons (Flash uniquement).
comp/example/programmingas3/podcastplayer/PlayButtonRenderer.as	Composant de rendu de cellule personnalisé permettant d'afficher un bouton de lecture dans une cellule de la grille de données (Flash uniquement).
com/example/programmingas3/podcastplayer/RSSBase.as	Une classe de base qui fournit les méthodes et les propriétés courantes pour la classe RSSChannel et la classe RSSItem.
com/example/programmingas3/podcastplayer/RSSChannel.as	Une classe ActionScript qui contient des données relatives à un canal RSS.
com/example/programmingas3/podcastplayer/RSSItem.as	Une classe ActionScript qui contient des données relatives à un élément RSS.
com/example/programmingas3/podcastplayer/SoundFacade.as	La classe ActionScript principale pour l'application. Elle encapsule les méthodes et les événements des classes Sound et SoundChannel et ajoute une prise en charge pour l'interruption et la reprise de la lecture.
com/example/programmingas3/podcastplayer/URLService.as	Une classe ActionScript qui récupère des données d'une URL distante.
playerconfig.xml	Un fichier XML contenant une liste des fils de syndication qui représentent des chaînes de podcast.
comp/example/programmingas3/utils/DateUtil.as	Classe permettant le formatage rapide de la date (Flash uniquement).

Lecture de données RSS pour une chaîne de podcast

L'application Podcast Player commence par lire les informations concernant des chaînes de podcasts et leurs épisodes :

1. L'application commence par lire un fichier de configuration XML qui contient une liste des chaînes de podcast et affiche la liste des chaînes pour l'utilisateur.

2. Lorsque l'utilisateur sélectionne l'une des chaînes de podcast, il lit le flux RSS pour la chaîne et affiche une liste des épisodes de chaîne.

Cet exemple utilise la classe d'utilitaire URLLoader pour récupérer des données de texte depuis un emplacement distant ou un fichier local. L'application Podcast Player crée d'abord un objet URLLoader pour obtenir une liste des fils de syndication au format XML du fichier playerconfig.xml. Ensuite, lorsque l'utilisateur sélectionne un fil de syndication spécifique dans la liste, un nouvel objet URLLoader est créé pour lire les données RSS de l'URL de ce fil.

Simplification de la lecture et du chargement du son à l'aide de la classe SoundFacade

L'architecture audio ActionScript 3.0 est puissante mais complexe. Les applications nécessitant des fonctions de lecture et de chargement de son de base uniquement peuvent utiliser une classe masquant une partie de la complexité en fournissant un ensemble d'appels et d'événements plus simple. Dans l'univers des modèles de conception de logiciel, une telle classe est appelée *façade*.

La classe SoundFacade présente une seule interface permettant d'effectuer les tâches suivantes :

- Chargement de fichiers audio à l'aide d'un objet Sound, d'un objet SoundLoaderContext et d'une classe SoundMixer
- Lecture de fichiers audio à l'aide des objets Sound et SoundChannel
- Envoi d'événements de progression de la lecture
- Interruption et reprise de la lecture du son à l'aide des objets Sound et SoundChannel

La classe SoundFacade essaie d'offrir le meilleur de la fonctionnalité des classes de son ActionScript avec moins de complexité.

Le code suivant indique la déclaration de classe, les propriétés de classe et la méthode constructeur `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();
    }
}
```

La classe SoundFacade étend la classe EventDispatcher pour qu'elle puisse envoyer ses propres événements. Le code de classe déclare d'abord les propriétés pour un objet Sound et un objet SoundChannel. La classe stocke également la valeur de l'URL du fichier audio et une propriété `bufferTime` à utiliser lors de la lecture du son en continu. De plus, elle accepte des valeurs de paramètre booléennes qui affectent le comportement de lecture et de chargement :

- Le paramètre `autoLoad` indique à l'objet que le chargement du son doit commencer dès la création de cet objet.
- Le paramètre `autoPlay` indique que la lecture du son doit commencer dès qu'une quantité suffisante de données audio a été chargée. S'il s'agit d'un son diffusé en continu, la lecture commence dès qu'une quantité suffisante de données (comme spécifié par la propriété `bufferTime`) est chargée.
- Le paramètre `streaming` indique que ce fichier audio peut commencer la lecture avant la fin du chargement.

Le paramètre `bufferTime` prend la valeur -1 par défaut. Si la méthode constructeur détecte une valeur négative dans le paramètre `bufferTime`, elle définit la propriété `bufferTime` sur la valeur de `SoundMixer.bufferTime`. Ceci permet à l'application de prendre la valeur `SoundMixer.bufferTime` globale, par défaut, comme souhaité.

Si le paramètre `autoLoad` est défini sur `true`, la méthode constructeur appelle immédiatement la méthode `load()` suivante pour commencer le chargement du fichier audio:

```
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);
}
```

La méthode `load()` crée un objet Sound puis ajoute des écouteurs pour tous les événements de son importants. Elle indique ensuite à l'objet Sound de charger le fichier audio, à l'aide d'un objet LoaderContext pour transmettre la valeur `bufferTime`.

Etant donné que la propriété `url` peut être modifiée, vous pouvez utiliser une occurrence de SoundFacade pour lire différents fichiers audio à la suite : il vous suffit de modifier la propriété `url` et d'appeler la méthode `load()` afin de charger le nouveau fichier audio.

Les trois méthodes d'écouteur d'événement suivantes indiquent comment l'objet SoundFacade suit la progression du chargement et décide quand lancer la lecture du son :

```
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();
    }
}
```

La méthode `onLoadOpen()` s'exécute lorsque le chargement du son commence. Si vous pouvez lire le son en mode continu, la méthode `onLoadComplete()` définit immédiatement l'indicateur `isReadyToPlay` sur `true`. L'indicateur `isReadyToPlay` détermine si l'application peut lancer la lecture du son, peut-être en réponse à une action utilisateur (clic sur un bouton de lecture, par exemple). La classe SoundChannel gère la mise en mémoire tampon des données audio. Par conséquent, il est inutile de vérifier si suffisamment de données ont été chargées avant d'appeler la méthode `play()`.

La méthode `onLoadProgress()` s'exécute régulièrement pendant le chargement. Elle envoie simplement une copie de son objet ProgressEvent pour le code qui utilise cet objet SoundFacade.

Une fois que les données audio ont été complètement chargées, la méthode `onLoadComplete()` s'exécute en appelant la méthode `play()` pour des sons non diffusés en continu, si nécessaire. La méthode `play()` est décrite ci-dessous.

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

La méthode `play()` appelle la méthode `Sound.play()` lorsque le son peut être lu. L'objet SoundChannel résultant est stocké dans la propriété `sc`. La méthode `play()` crée ensuite un objet Timer qui sera utilisé pour envoyer des événements de progression de la lecture à des intervalles réguliers.

Affichage de la progression de la lecture

La création d'un objet Timer pour surveiller la lecture est une opération complexe que vous devez coder une seule fois. Le fait d'encapsuler cette logique Timer dans une classe réutilisable telle que la classe SoundFacade permet aux applications d'écouter les mêmes types d'événements de progression lorsqu'un son est chargé et lorsqu'il est lu.

L'objet Timer créé par la méthode `SoundFacade.play()` envoie une occurrence de TimerEvent toutes les secondes. La méthode `onPlayTimer()` s'exécute chaque fois qu'un nouveau TimerEvent arrive :

```
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);
}
```

La méthode `onPlayTimer()` implémente la technique d'estimation de la taille décrite dans la section « Surveillance de la lecture » à la page 587. Elle crée ensuite une occurrence de ProgressEvent avec un type d'événement de `SoundFacade.PLAY_PROGRESS`, avec la propriété `bytesLoaded` définie sur la position actuelle de l'objet SoundChannel et la propriété `bytesTotal` définie sur la longueur estimée des données audio.

Interruption et reprise de la lecture

La méthode `SoundFacade.play()` décrite précédemment accepte un paramètre `pos` correspondant à une position de début dans les données audio. Si la valeur `pos` est zéro, la lecture du son commence au début.

La méthode `SoundFacade.stop()` accepte également un paramètre `pos`, comme indiqué ici :

```
public function stop(pos:int = 0):void
{
    if (this.isPlaying)
    {
        this.pausePosition = pos;
        this.sc.stop();
        this.playTimer.stop();
        this.isPlaying = false;
    }
}
```

Chaque fois que la méthode `SoundFacade.stop()` est appelée, elle définit la propriété `pausePosition` de façon à ce que l'application sache où positionner la tête de lecture si l'utilisateur souhaite reprendre la lecture du même son.

Les méthodes `SoundFacade.pause()` et `SoundFacade.resume()` indiquées ci-dessous appellent les méthodes `SoundFacade.stop()` et `SoundFacade.play()` respectivement, transmettant chaque fois une valeur de paramètre `pos`.

```
public function pause():void
{
    stop(this.sc.position);
}

public function resume():void
{
    play(this.pausePosition);
}
```

La méthode `pause()` transmet la valeur `SoundChannel.position` actuelle à la méthode `play()`, qui la stocke dans la propriété `pausePosition`. La méthode `resume()` recommence à lire le même son en utilisant la valeur `pausePosition` comme point de début.

Extension de l'exemple Podcast Player

Cet exemple présente un Podcast Player dépouillé qui présente l'utilisation de la classe SoundFacade réutilisable. Vous pouvez ajouter d'autres fonctions pour améliorer l'utilité de cette application, notamment :

- stocker la liste des fils de syndication et des informations d'utilisation concernant chaque épisode dans une occurrence de SharedObject pouvant être utilisée la prochaine fois que l'utilisateur exécute l'application ;
- permettre à l'utilisateur d'ajouter son fil de syndication à la liste des chaînes de podcast ;
- mémoriser la position de la tête de lecture lorsque l'utilisateur arrête ou quitte un épisode de façon à ce qu'il puisse être redémarré à partir de ce point la prochaine fois que l'utilisateur exécute l'application ;
- télécharger des fichiers mp3 d'épisodes pour les écouter hors ligne, lorsque l'utilisateur n'est pas connecté à Internet ;
- ajouter des fonctions d'abonnement qui vérifient régulièrement la présence de nouveaux épisodes dans une chaîne de podcast et mettre à jour la liste des épisodes automatiquement ;
- ajouter une fonctionnalité de recherche de podcasts à l'aide d'une API à partir d'un service d'hébergement de podcasts, tel que Odeo.com.

Chapitre 26 : Capture des données saisies par l'utilisateur

Ce chapitre décrit comment créer une interactivité à l'aide d'ActionScript 3.0 pour répondre à l'activité de l'utilisateur. Il traite des événements de souris et de clavier puis aborde des rubriques plus avancées, notamment la personnalisation du menu contextuel et la gestion du focus. Ce chapitre se termine par WordSearch, un exemple d'application qui répond aux actions de la souris.

Ce chapitre suppose que vous maîtrisez le modèle d'événement d'ActionScript 3.0. Pour plus d'informations, consultez le chapitre « Gestion des événements » à la page 254.

Principes de base de la saisie utilisateur

Introduction à la capture des données saisies par l'utilisateur

L'interaction utilisateur, au moyen du clavier, de la souris, de la caméra ou d'une combinaison de ces périphériques, est à la base de l'interactivité. Dans ActionScript 3.0, l'identification et la réponse à l'interaction utilisateur impliquent principalement l'écoute d'événements.

La classe InteractiveObject, une sous-classe de la classe DisplayObject, fournit la structure d'événements courante et la fonctionnalité nécessaire à la gestion de l'interaction utilisateur. Vous ne créez pas directement une occurrence de la classe InteractiveObject. Affichez plutôt des objets tels SimpleButton, Sprite, TextField et ainsi des composants divers de l'outil de programmation Flash et de Flex héritent leur modèle d'interaction utilisateur à partir de cette classe. Ils partagent alors une structure commune. Cela signifie que les techniques que vous apprenez et le code que vous écrivez pour gérer l'interaction utilisateur dans un objet dérivé de InteractiveObject sont applicables à tous les autres.

Les tâches d'interaction utilisateur classiques suivantes sont décrites dans ce chapitre :

- Capture de la saisie au clavier à l'échelle de l'application
- Capture de la saisie au clavier sur un objet d'affichage spécifique
- Capture des actions de la souris à l'échelle de l'application
- Capture des actions de la souris sur un objet d'affichage spécifique
- Création d'interactivité aux actions de glisser-déposer
- Création d'un curseur de souris personnalisé (pointeur de la souris)
- Ajout de nouveaux comportements au menu contextuel
- Gestion du focus

Concepts importants et terminologie

Il est important que vous vous familiarisiez avec les termes d'interaction utilisateur suivants avant de poursuivre :

- Code de caractère : code numérique représentant un caractère dans le jeu de caractères actuel (associé à une touche tapée sur le clavier). Par exemple, D et d ont des codes de caractères différents même si elles sont créées par la même touche sur un clavier anglais américain.

- Menu contextuel : menu qui apparaît lorsqu'un utilisateur clique avec le bouton droit de la souris ou utilise une combinaison clavier-souris particulière. Les commandes de menu contextuel s'appliquent généralement directement à l'élément sur lequel vous avez cliqué. Par exemple, un menu contextuel pour une image peut contenir une commande pour afficher l'image dans une fenêtre séparée et une commande pour la télécharger.
- Focus : indication qu'un élément sélectionné est actif et qu'il est la cible d'une interaction clavier ou souris.
- Code de touche : code numérique correspondant à une touche physique du clavier.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester ses exemples de code. Etant donné que ce chapitre traite de l'utilisation des données saisies par l'utilisateur dans ActionScript, pratiquement tous les exemples de code qu'il contient impliquent la manipulation d'un certain type d'objet d'affichage (généralement un champ de texte ou une sous-classe InteractiveObject). Dans le cadre de ces exemples, l'objet d'affichage peut avoir été créé et placé sur la scène dans Adobe® Flash® CS4 Professional ou bien créé à l'aide ActionScript. Le test d'un exemple implique l'affichage du résultat dans Flash Player ou or Adobe® AIR™ et l'interaction avec l'exemple pour visualiser les effets du code.

Pour tester les codes de ce chapitre :

1. Créez un document vide à l'aide de l'outil de programmation Flash.
2. Sélectionnez une image-clé dans le scénario.
3. Ouvrez le panneau Actions et copiez le code dans le panneau Script.
4. Créez une occurrence sur la scène :
   - Si le code fait référence à un champ de texte, utilisez l'outil Texte pour créer un champ de texte dynamique sur la scène.
   - Autrement, créez une occurrence de symbole de clip ou de bouton sur la scène.
5. Sélectionnez le champ de texte, le bouton ou le clip et attribuez-lui un nom d'occurrence dans l'Inspecteur des Propriétés. Le nom doit correspondre au nom de l'objet d'affichage dans l'exemple de code (par exemple, si le code manipule un objet appelé `myDisplayObject`, nommez votre objet scène `myDisplayObject` également).
6. Exécutez le programme en sélectionnant Contrôle > Tester l'animation.

   A l'écran, l'objet d'affichage est manipulé comme indiqué dans le code.

Capture de la saisie au clavier

Les objets d'affichage qui héritent de leur modèle d'interaction de la classe InteractiveObject peuvent répondre à des événements de clavier à l'aide d'écouteurs d'événement. Par exemple, vous pouvez placer un écouteur d'événement sur la scène pour écouter et répondre à une saisie au clavier. Dans le code suivant, un écouteur d'événement capture une saisie au clavier et le nom et les propriétés de code de la touche sont affichés :

```
function reportKeyDown(event:KeyboardEvent):void
{
    trace("Key Pressed: " + String.fromCharCode(event.charCode) + " (character code: " +
event.charCode + ")");
}
stage.addEventListener(KeyboardEvent.KEY_DOWN, reportKeyDown);
```

Certaines touches (Ctrl, par exemple) génèrent des événements, même si elles n'ont pas de représentation de glyphes.

Dans l'exemple de code précédent, l'écouteur d'événement de clavier a capturé une saisie au clavier pour la scène entière. Vous pouvez également écrire un écouteur d'événement pour un objet d'affichage spécifique sur la scène ; cet écouteur d'événement est déclenché lorsque l'objet a le focus.

Dans l'exemple suivant, les frappes de touches apparaissent dans le panneau Sortie uniquement lorsque l'utilisateur tape dans l'occurrence de TextField. S'il maintient la touche Maj enfoncée, la couleur du contour du TextField devient temporairement rouge.

Ce code suppose qu'il existe une occurrence de TextField appelée `tf` sur la scène.

```
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;
    }
}
```

La classe TextField signale également un événement `textInput` que vous pouvez écouter lorsqu'un utilisateur saisit du texte. Pour plus d'informations, consultez la section « Capture du texte saisi par l'utilisateur » à la page 450.

Présentation des codes de touches et de caractères

Les propriétés `keyCode` et `charCode` d'un événement clavier permettent de déterminer la touche utilisée et de déclencher d'autres actions. La propriété`keyCode` est une valeur numérique qui correspond à la valeur de la touche sur le clavier. La propriété `charCode` est la valeur numérique de cette touche dans le jeu de caractères actuel. (Le jeu de caractères par défaut est UTF-8, qui prend en charge ASCII.)

La différence principale entre le code de touche et les valeurs de caractères est la suivante : la valeur du code de touche représente une touche déterminée du clavier (la touche 1 sur le pavé numérique est différente du 1 sur le clavier central, mais cette dernière permet à la fois de générer 1 et &) alors que la valeur du caractère représente un caractère particulier (les caractères R et r sont différents).

Remarque : *pour la concordance entre les touches et les codes de caractère ASCII correspondants, reportez-vous à la classe flash.ui.Keyboard dans le Guide de référence du langage ActionScript.*

Les mappages entre les touches et leurs codes de touches dépendent du périphérique et du système d'exploitation. C'est pourquoi vous ne devez pas utiliser de mappages de touches pour déclencher des actions. A la place, utilisez les valeurs de constante prédéfinies fournies par la classe Keyboard pour référencer les propriétés `keyCode` appropriées. Par exemple, au lieu d'utiliser le mappage de touche pour la touche Maj, utilisez la constante `Keyboard.SHIFT` (comme indiqué dans l'exemple de code précédent).

Présentation de la priorité de KeyboardEvent

Comme avec d'autres événements, la séquence d'événement de clavier est déterminée par la hiérarchie d'objet d'affichage et non par l'ordre dans lequel les méthodes `addEventListener()` sont affectées dans le code.

Supposons par exemple que vous placiez un champ de texte `tf` au sein d'un clip nommé `container` et que vous ajoutiez un écouteur d'événement pour l'événement de clavier à chaque occurrence, comme indiqué dans l'exemple suivant :

```
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 + ")");
}
```

Etant donné qu'il existe un écouteur sur le champ de texte et sur son conteneur parent, la fonction `reportKeyDown()` est appelée deux fois pour chaque frappe de touche dans le TextField. Notez que pour chaque touche actionnée, le champ de texte envoie un événement avant que le clip `container` ne distribue un événement.

Le système d'exploitation et le navigateur Web traiteront les événements de clavier avant Adobe Flash Player ou AIR. Par exemple, dans Microsoft Internet Explorer, lorsque vous appuyez sur Ctrl+W, vous fermez la fenêtre du navigateur avant qu'un fichier SWF contenu ne distribue un événement de clavier.

Capture des entrées de souris

Les clics de souris créent des événements souris qui permettent de déclencher une fonctionnalité interactive. Il est possible d'ajouter un écouteur d'événement à la scène afin d'écouter les événements de souris qui se produisent à tout endroit du fichier SWF. Vous pouvez également ajouter des écouteurs d'événement à des objets sur la scène qui héritent de InteractiveObject (par exemple, Sprite ou MovieClip) ; ces écouteurs sont déclenchés lorsque vous cliquez sur l'objet.

Comme les événements de clavier, les événements de souris se propagent vers le haut. Dans l'exemple suivant, `square` étant un enfant de la scène, l'événement est distribué à la fois du sprite `square` et de l'objet scène en cas de clic sur le carré :

```
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 + "]");
}
```

Dans l'exemple précédent, notez que l'événement de souris contient des informations sur l'endroit où le clic s'est produit. Les propriétés `localX` et `localY` indiquent l'emplacement du clic sur l'enfant le plus bas de la chaîne d'affichage. Par exemple, si vous cliquez dans l'angle supérieur gauche de `square`, les coordonnées locales [0,0] sont signalées car il s'agit du point d'alignement de `square`. Autrement, les propriétés `stageX` et `stageY` se réfèrent aux coordonnées globales du clic sur la scène. Le même clic signale [50,50] pour ces coordonnées car `square` y a été déplacé. Ces deux paires de coordonnées peuvent être utiles, selon la façon dont vous souhaitez répondre à l'interaction utilisateur.

L'objet MouseEvent contient aussi les propriétés booléennes `altKey`, `ctrlKey` et `shiftKey`. Vous pouvez utiliser ces propriétés pour vérifier si l'utilisateur appuie également sur la touche Alt, Ctrl ou Maj au moment du clic de la souris.

Création de fonctionnalité glisser-déposer

La fonctionnalité glisser-déposer permet aux utilisateurs de sélectionner un objet tout en maintenant le bouton gauche de la souris enfoncé, de déplacer l'objet à un nouvel endroit sur l'écran et de l'y déposer en relâchant le bouton gauche de la souris. Le code suivant en est un exemple :

```
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);
}
```

Pour plus de détails, consultez la section sur la création d'une interaction glisser-déposer dans « Modification de la position » à la page 299.

Personnalisation du curseur de la souris

Le curseur de la souris (pointeur de la souris) peut être masqué ou remplacé pour tout objet d'affichage de la scène. Pour masquer ce curseur, appelez la méthode `Mouse.hide()`. Personnalisez le curseur en appelant `Mouse.hide()`, en écoutant l'événement `MouseEvent.MOUSE_MOVE` en provenance de la scène et en définissant les coordonnées d'un objet d'affichage (votre curseur personnalisé) sur les propriétés `stageX` et `stageY` de l'événement. L'exemple suivant illustre une exécution de base de cette tâche :

```
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;
}
```

Personnalisation du menu contextuel

Chaque objet issu de la classe InteractiveObject peut posséder un menu contextuel exclusif, qui s'affiche lorsque l'utilisateur clique avec le bouton droit de la souris dans le fichier SWF. Plusieurs commandes sont incluses par défaut, notamment des commandes d'avance, de recul, d'impression, de qualité et de zoom.

Vous pouvez supprimer toutes les commandes par défaut du menu, à l'exception des commandes Paramètres et A propos. Lorsque vous définissez la propriété Stage `showDefaultContextMenu` sur `false`, ces commandes sont supprimées du menu contextuel.

Pour créer un menu contextuel personnalisé pour un objet d'affichage particulier, créez une occurrence de la classe ContextMenu, appelez la méthode `hideBuiltInItems()` et affectez cette occurrence à la propriété `contextMenu` de cette occurrence de DisplayObject. L'exemple suivant crée un carré dessiné dynamiquement avec une commande de menu contextuel qui permet de modifier sa couleur au hasard :

```
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);
}
```

Gestion du focus

Un objet interactif peut recevoir le focus, soit par programmation, soit par le biais d'une action utilisateur. Dans les deux cas, la définition du focus règle la propriété `focus` de l'objet sur `true`. En outre, si la propriété `tabEnabled` a la valeur `true`, l'utilisateur peut transmettre le focus d'un objet à un autre en appuyant sur la touche Tabulation. La valeur `tabEnabled` est `false` par défaut, excepté dans les cas suivants :

- Pour un objet SimpleButton, la valeur est `true`.
- Pour un champ de texte d'entrée, la valeur est `true`.
- Pour un objet Sprite ou MovieClip dont `buttonMode` a la valeur `true`, la valeur est `true`.

Dans chacune de ces situations, vous pouvez ajouter un écouteur pour `FocusEvent.FOCUS_IN` ou `FocusEvent.FOCUS_OUT` pour étendre les comportements possibles lors du changement de focus. Si cette technique est pratique pour les champs texte et les formulaires, vous pouvez aussi l'utiliser avec les sprites, les clips et tout autre objet qui hérite de la classe InteractiveObject. L'exemple suivant montre comment activer le changement de focus avec la touche de tabulation et comment répondre à l'événement focus qui en découle. Dans ce cas, chaque carré change de couleur lorsqu'il reçoit le focus.

Remarque : *l'outil de programmation Flash utilise des raccourcis clavier pour gérer le focus. Par conséquent, pour simuler correctement la gestion du focus, les fichiers SWF doivent être testés dans un navigateur ou dans AIR plutôt que dans Flash.*

```
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);
}
```

Exemple : WordSearch

Cet exemple illustre l'interaction utilisateur en gérant des événements de souris. Les utilisateurs créent autant de mots que possible à partir d'une grille aléatoire de lettres, en se déplaçant horizontalement ou verticalement dans la grille, mais en n'utilisant jamais deux fois la même lettre. Cet exemple illustre les fonctions suivantes d'ActionScript 3.0 :

- Elaboration dynamique d'une grille de composants
- Réponse aux événements souris
- Suivi du score en fonction de l'interaction utilisateur

Pour obtenir les fichiers d'application associés à cet exemple, voir www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application WordSearch se trouvent dans le dossier Samples/WordSearch. L'application se compose des fichiers suivants :

Fichier	Description
WordSearch.as	Classe comportant les principales fonctionnalités de l'application.
WordSearch.fla ou WordSearch.mxml	Le fichier d'application principal pour Flex (MXML) ou Flash (FLA).
dictionary.txt	Un fichier utilisé pour déterminer si les mots sont écrits correctement.

Chargement d'un dictionnaire

Pour créer un jeu qui implique la recherche de mots, un dictionnaire est nécessaire. L'exemple inclut un fichier texte appelé dictionary.txt, qui contient une liste de mots séparés par des retours à la ligne. Après avoir créé un tableau appelé `words`, la fonction `loadDictionary()` demande ce fichier. Une fois chargé, celui-ci devient une longue chaîne. Vous pouvez convertir cette chaîne en un tableau de mots grâce à la méthode `split()`, qui s'arrête à chaque occurrence d'un retour à la ligne (code de caractère 10) ou d'une nouvelle ligne (code de caractère 13). Cette analyse a lieu dans la fonction `dictionaryLoaded()` :

```
words = dictionaryText.split(String.fromCharCode(13, 10));
```

Création de l'interface utilisateur

Une fois les mots stockés, vous pouvez élaborer l'interface utilisateur. Créez deux occurrences de bouton : l'une permet de soumettre un mot, l'autre d'effacer le mot en cours de saisie. Dans chaque cas, vous devez répondre à la saisie utilisateur en écoutant l'événement `MouseEvent.CLICK` que le bouton diffuse et en appelant ensuite une fonction. Dans la fonction `setupUI()`, ce code crée les écouteurs sur les deux boutons :

```
submitWordButton.addEventListener(MouseEvent.CLICK,submitWord);
clearWordButton.addEventListener(MouseEvent.CLICK,clearWord);
```

Génération d'un tableau de jeu

Le tableau de jeu est une grille de lettres aléatoires. Dans la fonction `generateBoard()`, une grille en deux dimensions est créée en imbriquant une boucle dans une autre. La première boucle incrémente les lignes et la seconde le nombre totale de colonnes par ligne. Chaque cellule créée par ces lignes et ces colonnes contient un bouton qui représente une lettre sur le tableau.

```
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);
        }
    }
}
```

Même si un écouteur est ajouté pour un événement `MouseEvent.CLICK` sur une seule ligne (car il est dans une boucle `for`), il est affecté à chaque occurrence de bouton. Chaque bouton reçoit un nom dérivé de la position de sa ligne et de sa colonne, ce qui permet de référencer facilement la ligne et la colonne de chaque bouton ultérieurement dans le code.

Création de mots à partir de la saisie utilisateur

Les mots peuvent être écrits en sélectionnant des lettres adjacentes verticalement ou horizontalement, mais jamais en utilisant deux fois la même lettre. Chaque clic génère un événement de souris au niveau duquel le mot qu'écrit l'utilisateur doit être vérifié pour s'assurer qu'il se poursuit correctement à partir de lettres cliquées précédemment. Si ce n'est pas le cas, le mot précédent est supprimé et un nouveau est commencé. Cette vérification s'effectue dans la méthode `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));
}
```

Les méthodes `charAt()` et `indexOf()` de la classe String récupèrent les lignes et les colonnes du bouton sur lequel l'utilisateur clique actuellement et celui sur lequel il vient de cliquer. La méthode `isLegalContinuation()` renvoie `true` si la ligne ou la colonne est la même et que la ligne ou la colonne qui a changé se trouve à un seul incrément de la précédente. Si vous souhaitez modifier les règles du jeu et autoriser une lecture diagonale, vous pouvez supprimer la vérification de la ligne ou de la colonne inchangée. Dans ce cas, la ligne finale serait la suivante :

```
return (Math.abs(prevButtonRow - currButtonRow) <= 1) && Math.abs(prevButtonCol -
currButtonCol) <= 1));
```

Vérification des mots soumis

Pour terminer le code pour le jeu, des mécanismes permettant de vérifier des mots soumis et de gérer le score sont nécessaires. La méthode `searchForWord()` permet ces deux opérations :

```
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;
}
```

Cette fonction analyse en boucle tous les mots du dictionnaire. Si le mot de l'utilisateur correspond à un mot du dictionnaire, sa position dans le dictionnaire est renvoyée. La méthode `submitWord()` vérifie la réponse et met à jour le score si la position est valide.

Personnalisation

Il existe plusieurs constantes au début de la classe. Vous pouvez modifier ce jeu en changeant ces variables. Il est par exemple possible de modifier le temps de jeu disponible en augmentant la variable `TOTAL_TIME`. Si vous augmentez légèrement la variable `PERCENT_VOWELS`, vous pouvez accroître la probabilité de trouver des mots.

Chapitre 27 : Mise en réseau et techniques de communication

Ce chapitre explique comment permettre à votre fichier SWF de communiquer avec des fichiers externes et d'autres occurrences d'Adobe Flash Player et d'Adobe AIR. Il décrit également comment charger des données à partir de sources externes, envoyer des messages entre le serveur Java et Flash Player et effectuer des chargements et téléchargements de fichier à l'aide des classes FileReference et FileReferenceList.

Principes de base de la mise en réseau et de la communication

Introduction à la mise en réseau et à la communication

L'élaboration d'applications ActionScript plus complexes vous oblige souvent à communiquer avec des scripts côté serveur ou à charger des données à partir de fichier XML ou texte externes. Le package flash.net contient des classes qui permettent d'envoyer et de recevoir des données sur Internet (par exemple, pour charger du contenu à partir d'une URL distante, communiquer avec d'autres occurrences de Flash Player ou d'AIR et se connecter à des sites Web distants).

Dans ActionScript 3.0, ce sont les classes URLLoader et URLRequest qui permettent de le faire. Vous utilisez ensuite une classe spécifique pour accéder aux données, selon le type de données chargées. Par exemple, si le contenu distant se présente sous forme de paires nom-valeur, vous pouvez utiliser la classe URLVariables pour analyser les résultats du serveur. Autrement, si le fichier chargé à l'aide des classes URLLoader et URLRequest est un document XML distant, vous pouvez l'analyser au moyen du constructeur de la classe XML, de celui de la classe XMLDocument ou de la méthode `XMLDocument.parseXML()`. Cela vous permet de simplifier votre code ActionScript puisque le code utilisé pour le chargement de fichiers externes est le même, que vous utilisiez les classes URLVariables, XML ou toute autre classe pour analyser et utiliser les données distantes.

Le package flash.net contient également des classes pour d'autres types de communication distante. Elles comprennent la classe FileReference pour le téléchargement des fichiers depuis un serveur, les classes Socket et XMLSocket qui permettent de communiquer directement avec des ordinateurs distants à l'aide de connexions socket, et les classes NetConnection et NetStream utilisées pour communiquer avec des ressources serveur propres à Flash (serveurs Flash Media Server et Flash Remoting, par exemple) et pour charger des fichiers vidéo.

Pour finir, le package flash.net contient des classes pour la communication sur l'ordinateur local de l'utilisateur. Celles-ci comprennent la classe LocalConnection, qui permet de communiquer entre deux ou plusieurs fichiers SWF exécutés sur un seul ordinateur, et la classe SharedObject, qui permet de stocker des données sur l'ordinateur d'un utilisateur et de les récupérer ultérieurement lorsqu'elles sont renvoyées à votre application.

Tâches courantes de mise en réseau et de communication

La liste suivante décrit les tâches les plus courantes que vous souhaiterez effectuer concernant la communication externe à partir d'ActionScript. Elles sont traitées dans ce chapitre :

- Chargement de données d'un fichier externe ou d'un script serveur
- Envoi de données à un script serveur

- Communication avec d'autres fichiers SWF locaux
- Utilisation de connexions socket binaires
- Communication avec des sockets XML
- Stockage des données locales persistantes
- Chargement de fichiers sur un serveur
- Chargement de fichiers d'un serveur à la machine de l'utilisateur

Concepts importants et terminologie

La liste de référence suivante énumère les termes importants que vous rencontrerez dans ce chapitre :

- Données externes : données stockées sous une certaine forme en dehors du fichier SWF et téléchargées dans ce dernier si besoin est. Ces données peuvent être stockées dans un fichier téléchargé directement, dans une base de données ou sous une autre forme récupérée en appelant des scripts ou des programmes exécutés sur un serveur.
- Variables codées URL : le format codé URL permet de représenter plusieurs variables (paires de valeurs et de noms de variable) dans une seule chaîne de texte. Les variables individuelles sont écrites dans le format `name=value`. Chaque variable (c'est-à-dire chaque paire nom-valeur) est séparée par des caractères esperluettes, de la façon suivante : `variable1=value1&variable2=value2` De cette façon, un nombre infini de variables peut être envoyé sous la forme d'un seul message.
- Type MIME : code standard utilisé pour identifier le type d'un fichier donné dans une communication Internet. Tous les types de fichier ont un code spécifique utilisé pour les identifier. Lors de l'envoi d'un fichier ou d'un message, un ordinateur (un serveur Web ou l'occurrence Flash Player ou AIR d'un utilisateur, par exemple) spécifie le type de fichier envoyé.
- HTTP : Hypertext Transfer Protocol—format standard de livraison de pages Web et de différents types de contenu envoyés sur Internet.
- Méthode de requête : lorsqu'un programme tel que Flash Player ou un navigateur Web envoie un message (appelé requête HTTP) à un serveur Web, les données envoyées peuvent être intégrées dans la requête de deux façons différentes : les *méthodes de requêtes* GET et POST. Côté serveur, le programme qui reçoit la requête doit pouvoir rechercher les données dans la portion appropriée de la requête. C'est pourquoi la méthode de requête utilisée pour envoyer des données d'ActionScript doit correspondre à celle utilisée pour lire ces données sur le serveur.
- Connexion socket : une connexion permanente pour la communication entre deux ordinateurs.
- Charger : envoyer un fichier à un autre ordinateur.
- Télécharger : récupérer un fichier d'un autre ordinateur.

Utilisation d'adresses IPv6

Flash Player 9.0.115.0 (et versions ultérieures) prennent en charge IPv6 (Internet Protocol version 6). IPv6 est une version du protocole IP (Internet Protocol) qui prend en charge les adresses 128 bits (amélioration du protocole IPv4 précédent qui prend en charge les adresses 32 bits). Vous devrez peut-être activer IPv6 sur vos interfaces de mise en réseau. Pour plus d'informations, consultez l'Aide du système d'exploitation hébergeant les données.

Si IPv6 est pris en charge sur le système hébergeant, vous pouvez spécifier des adresses littérales IPv6 numériques dans les URL entre crochets ([]), comme suit :

```
rtmp://[2001:db8:ccc3:ffff:0:444d:555e:666f]:1935/test
```

Flash Player renvoie les valeurs IPv6 littérales selon les règles suivantes :

- Flash Player renvoie la forme complète de la chaîne pour les adresses IPv6.
- La valeur IP ne possède pas d'abréviations à deux-points redoublés (::).
- Les chiffres hexadécimaux sont en bas de casse seulement.
- Les adresses IPv6 sont entourées de crochets ([]).
- Chaque quatuor d'adresses est représenté par 0 à 4 chiffres hexadécimaux, sans zéros non significatifs.
- Un quatuor d'adresses de zéros est représenté par un seul zéro (et pas un caractère de deux-points redoublé), sauf dans les cas suivants.

Les valeurs IPv6 que renvoie Flash Player sont soumises aux exceptions suivantes :

- Une adresse IPv6 non spécifiée (rien que des zéros) est représentée par [::].
- L'adresse de bouclage (loopback) ou localhost IPv6 est représentée par [::1].
- Les adresses mappées en IPv4 (converties en IPv6) sont représentées par [::ffff:a.b.c.d], où a.b.c.d constitue une valeur décimale à points (dotted-decimal) IPv4 classique.
- Les adresses compatibles avec IPv4 sont représentées par [::a.b.c.d], où a.b.c.d est une valeur décimale à points (dotted-decimal) IPv4 classique.

Utilisation des exemples fournis dans ce chapitre

Vous pouvez profiter de la lecture de ce chapitre pour tester des exemples de programmes. Plusieurs de ces programmes chargent des données externes ou exécutent d'autres types de communications ; ces exemples contiennent souvent des appels de la fonction `trace()` de telle sorte que les résultats de l'exécution de l'exemple sont affichés dans le panneau Sortie. D'autres exemples effectuent une fonction telle que le chargement d'un fichier sur un serveur. Le test de ces exemples impliquera l'interaction avec le SWF et la confirmation qu'ils exécutent l'action attendue.

Les exemples de code peuvent être classés en deux catégories. Certains exemples sont écrits en supposant que le code est un script autonome (lié à une image-clé dans un document Flash, par exemple). Pour tester ces exemples :

1. Créez un nouveau document Flash.
2. Sélectionnez l'image-clé sur l'image 1 du scénario puis ouvrez le panneau Actions.
3. Copiez le code dans le panneau Script.
4. Dans le menu principal, choisissez Contrôle > Tester l'animation pour créer le fichier SWF et tester l'exemple.

D'autres exemples de code sont rédigés en tant que classe ; on s'attend à ce que la classe de l'exemple serve de classe de document pour le document Flash. Pour tester ces exemples :

1. Créez un document Flash vide et enregistrez-le sur votre ordinateur
2. Créez un nouveau fichier ActionScript et enregistrez-le dans le même répertoire que le document Flash. Le nom du fichier doit correspondre au nom de la classe du code. Par exemple, si le code définit une classe appelée “UploadTest”, enregistrez le fichier ActionScript sous le nom de “UploadTest.as”.
3. Copiez le code dans le fichier ActionScript et enregistrez le fichier.
4. Dans le document Flash, cliquez sur une partie vide de la scène ou de l'espace de travail pour activer l'Inspecteur des propriétés du document.
5. Dans l'Inspecteur des propriétés, dans le champ Classe du document, saisissez le nom de la classe ActionScript que vous avez copiée du texte.

6 Exécutez le programme en sélectionnant Contrôle > Tester l'animation et testez l'exemple.

Enfin, certains exemples fournis dans ce chapitre impliquent l'interaction avec un programme exécuté sur un serveur. Ces exemples contiennent le code qui peut être utilisé pour créer le programme nécessaire au test de l'exemple ; il vous faudra mettre en oeuvre les applications appropriées sur un serveur Web pour entreprendre ce test.

Utilisation de données externes

ActionScript 3.0 comprend des mécanismes permettant de charger des données depuis des sources externes. Ces sources peuvent être du contenu statique tel que des fichiers de texte ou du contenu dynamique tel qu'un script Web qui récupère des données d'une base de données. Les données peuvent être formatées de plusieurs façons, et ActionScript fournit une fonctionnalité pour décoder les données et y accéder. Vous pouvez également envoyer des données au serveur externe lors de leur récupération.

Utilisation des classes URLLoader et URLVariables

ActionScript 3.0 utilise les classes URLLoader et URLVariables pour charger des données externes. La classe URLLoader télécharge des données à partir d'une URL sous forme de texte, de données binaires ou de variables de code URL. La classe URLLoader est utile pour le téléchargement des fichiers texte et XML, ou d'autres informations destinées à des applications ActionScript dynamiques et orientées données. La classe URLLoader tire parti du modèle de gestion des événements d'ActionScript 3.0 qui permet d'écouter des événements tels que`complete`, `httpStatus`, `ioError`, `open`, `progress` et `securityError`. Le nouveau modèle de gestion des événements constitue une avancée considérable par rapport à ActionScript 2.0 pour les gestionnaires d'événements `LoadVars.onData`, `LoadVars.onHTTPStatus` et `LoadVars.onLoad` parce qu'il permet une gestion plus efficace des erreurs et des événements. Pour plus d'informations sur la gestion d'événements, consultez le chapitre « Gestion des événements » à la page 254.

Comme dans le cas des classes XML et LoadVars dans les versions antérieures d'ActionScript, les données obtenues par URLLoader ne sont disponibles qu'une fois le téléchargement terminé. Vous pouvez suivre la progression du téléchargement (nombre d'octets chargés et nombre total) en écoutant l'événement `flash.events.ProgressEvent.PROGRESS` qui doit être distribué. Toutefois, si le chargement du fichier s'effectue trop rapidement, il est possible que l'événement `ProgressEvent.PROGRESS` ne soit pas distribué. Une fois que le téléchargement d'un fichier est terminé, l'événement `flash.events.Event.COMPLETE` est distribué. Les données chargées sont décodées du format UTF-8 ou UTF-16 en chaînes.

Remarque : *si aucune valeur n'est définie pour `URLRequest.contentType`, les valeurs sont envoyées sous la forme `application/x-www-form-urlencoded`.*

La méthode `URLLoader.load()` (et éventuellement le constructeur de la classe URLLoader) prend un seul paramètre, `request`, qui correspond à une occurrence de URLRequest. Une occurrence de URLRequest contient toutes les informations d'une requête HTTP unique, telles que l'URL cible, la méthode de requête (`GET` ou `POST`), les informations d'en-tête supplémentaires et le type MIME (si vous chargez du contenu XML par exemple).

Pour charger un paquet XML dans un script côté serveur, par exemple, vous pouvez utiliser le code ActionScript 3.0 ci-après :

```
var secondsUTC:Number = new Date().time;
var dataXML:XML =
    <login>
        <time>{secondsUTC}</time>
        <username>Ernie</username>
        <password>guru</password>
    </login>;
var request:URLRequest = new URLRequest("http://www.yourdomain.com/login.cfm");
request.contentType = "text/xml";
request.data = dataXML.toXMLString();
request.method = URLRequestMethod.POST;
var loader:URLLoader = new URLLoader();
try
{
    loader.load(request);
}
catch (error:ArgumentError)
{
    trace("An ArgumentError has occurred.");
}
catch (error:SecurityError)
{
    trace("A SecurityError has occurred.");
}
```

L'extrait ci-dessus crée une occurrence de XML appelée `dataXML`, qui contient un paquet XML à envoyer au serveur. Ensuite, attribuez à la propriété `contentType` d'URLRequest la valeur `"text/xml"` et à la propriété `data` le contenu du paquet XML. Ces valeurs sont alors converties en chaînes à l'aide de la méthode `XML.toXMLString()`. Enfin, créez une nouvelle occurrence de URLLoader et envoyez la requête au script distant à l'aide de la méthode `URLLoader.load()`.

Vous pouvez spécifier les paramètres à transmettre dans la requête URL de trois manières :

- Au sein du constructeur URLVariables
- Au sein de la méthode `URLVariables.decode()`
- Sous forme d'une propriété particulière au sein de l'objet URLVariables lui-même

Lorsque vous définissez des variables au sein du constructeur URLVariables ou de la méthode `URLVariables.decode()`, vous devez veiller à encoder l'esperluette au format URL parce que ce caractère revêt un sens particulier et joue le rôle de délimiteur. Par exemple, lorsque vous transmettez une esperluette, vous devez l'encoder en remplaçant `&` par `%26`.

Chargement de données à partir de documents externes

Lorsque vous élaborez une application dynamique avec ActionScript 3.0, il est judicieux de charger les données à partir de fichiers externes ou de scripts côté serveur. Cela vous permet de construire vos applications dynamiques sans avoir à modifier ou recompiler vos fichiers ActionScript. Par exemple, si vous créez une application « conseil du jour », vous pouvez écrire un script côté serveur qui récupère un conseil au hasard dans une base de données et l'enregistre dans un fichier texte une fois par jour. Votre application ActionScript peut ensuite charger le contenu du fichier texte statique au lieu d'envoyer une requête à la base de données à chaque fois.

L'extrait de code suivant crée un objet URLRequest et URLLoader, qui charge le contenu d'un fichier texte externe nommé params.txt :

```
var request:URLRequest = new URLRequest("params.txt");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

Vous pouvez simplifier cet extrait de code de la manière suivante :

```
var loader:URLLoader = new URLLoader(new URLRequest("params.txt"));
```

Par défaut, si vous ne définissez aucune méthode de requête, Flash Player et AIR chargent le contenu à l'aide de la méthode HTTP `GET`. Si vous souhaitez envoyer des données avec la méthode `POST`, vous devez lui attribuer la propriété `request.method` à l'aide de la constante statique `URLRequestMethod.POST`, comme le montre le code suivant :

```
var request:URLRequest = new URLRequest("sendfeedback.cfm");
request.method = URLRequestMethod.POST;
```

Le document externe params.txt, chargé au moment de l'exécution, contient les données suivantes :

```
monthNames=January,February,March,April,May,June,July,August,September,October,November,Dece
mber&dayNames=Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Saturday
```

Le fichier contient deux paramètres, `monthNames` et `dayNames`. Chacun des paramètres contient une liste séparée par des virgules qui est analysée sous forme de chaîne. Vous pouvez transformer cette liste en tableau à l'aide de la méthode `String.split()`.

Evitez d'utiliser des mots réservés ou des éléments de langage comme noms de variables dans les fichiers de données externes, car cela complique la lecture et le débogage du code.

Une fois les données chargées, l'événement `Event.COMPLETE` est distribué et le contenu du document externe peut alors être utilisé dans la propriété `data` de URLLoader, comme l'illustre le code suivant :

```
private function completeHandler(event:Event):void
{
    var loader2:URLLoader = URLLoader(event.target);
    trace(loader2.data);
}
```

Si le document distant contient des paires nom-valeur, vous pouvez analyser les données à l'aide de la classe URLVariables en transmettant le contenu du fichier chargé, comme suit :

```
private function completeHandler(event:Event):void
{
    var loader2:URLLoader = URLLoader(event.target);
    var variables:URLVariables = new URLVariables(loader2.data);
    trace(variables.dayNames);
}
```

Chaque paire nom-valeur issue du fichier externe devient une nouvelle propriété de l'objet URLVariables. Chaque propriété de cet objet dans l'exemple de code précédent est traité comme une chaîne. Si la valeur de la paire nom-valeur est une liste d'éléments, vous pouvez convertir la chaîne en tableau en appelant la méthode `String.split()`, comme suit :

```
var dayNameArray:Array = variables.dayNames.split(",");
```

Si vous devez charger des données numériques à partir de fichiers externes, vous devez les convertir en valeurs numériques à l'aide d'une fonction de niveau supérieur, telle que `int()`, `uint()` ou `Number()`.

Au lieu de charger le contenu du fichier distant sous forme de chaîne et de créer un objet URLVariables, vous pouvez attribuer à la propriété `URLLoader.dataFormat` la valeur de l'une des propriétés statiques de la classe URLLoaderDataFormat. Les trois valeurs possibles de la propriété `URLLoader.dataFormat` sont les suivantes :

- `URLLoaderDataFormat.BINARY` : la propriété `URLLoader.data` contient des données binaires stockées dans un objet ByteArray.
- `URLLoaderDataFormat.TEXT` : la propriété `URLLoader.data` contient du texte sous forme d'objet String.
- `URLLoaderDataFormat.VARIABLES` : la propriété `URLLoader.data` contient des variables encodées au format URL issues d'un objet URLVariables.

Le code suivant montre comment vous pouvez automatiquement analyser les données chargées dans un objet URLVariables en attribuant à la propriété `URLLoader.dataFormat` la valeur `URLLoaderDataFormat.VARIABLES`.

```
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);
        }
    }
}
```

Remarque : *la valeur par défaut de `URLLoader.dataFormat` est `URLLoaderDataFormat.TEXT`.*

Comme le montre l'exemple suivant, le chargement de données XML à partir d'un fichier externe s'effectue comme le chargement de données URLVariables. Vous pouvez créer une occurrence de URLRequest et une occurrence de URLLoader et vous en servir pour télécharger un document XML distant. Lorsque le fichier est complètement téléchargé, l'événement `Event.COMPLETE` est distribué et le contenu du fichier externe est converti en occurrence de XML, que vous pouvez analyser avec les méthodes et propriétés XML.

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

Communication avec des scripts externes

Outre le chargement de fichiers de données externes, la classe URLVariables permet d'envoyer des variables à un script côté serveur et de traiter la réponse du serveur. Cela s'avère utile, par exemple, si vous programmez un jeu et souhaitez envoyer le score de l'utilisateur à un serveur afin de vérifier s'il faut l'ajouter à la liste des scores les plus élevés ; ou encore envoyer les informations de connexion de l'utilisateur au serveur pour validation. Un script côté serveur peut traiter le nom d'utilisateur et le mot de passe, les compare à une base de données et confirmer en retour la validité des informations fournies par l'utilisateur.

L'extrait de code ci-après crée un objet URLVariables nommé `variables`, qui génère une nouvelle variable appelée `name`. Ensuite, un objet URLRequest est créé pour spécifier l'URL du script côté serveur à laquelle doivent être envoyées les variables. Vous pouvez alors définir la propriété `method` de l'objet URLRequest pour envoyer les variables sous forme de requête HTTP `POST` . Pour ajouter l'objet URLVariables à la requête URL, définissez la propriété `data` de l'objet URLRequest sur l'objet URLVariables créé plus tôt. Enfin, l'occurrence de URLLoader est créée et la méthode `URLLoader.load()` appelée ; cette dernière lance la requête.

```
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);
}
```

Le code suivant reprend le contenu du document Adobe ColdFusion® greeting.cfm utilisé dans l'exemple précédent :

```
<cfif NOT IsDefined("Form.name") OR Len(Trim(Form.Name)) EQ 0>
    <cfset Form.Name = "Stranger" />
</cfif>
<cfoutput>welcomeMessage=#UrlEncodedFormat("Welcome, " & Form.name)#
</cfoutput>
```

Connexion à d'autres occurrences de Flash Player et d'AIR

La classe LocalConnection vous permet de communiquer avec différentes occurrences de Flash Player et d'AIR, par exemple un fichier SWF dans un conteneur HTML ou dans un lecteur intégré ou autonome. Vous pouvez ainsi élaborer des applications très polyvalentes, capables de partager des données entre plusieurs occurrences de Flash Player et d'AIR, par exemple des fichiers SWF exécutés dans un navigateur Web ou incorporés dans des applications de bureau.

La classe LocalConnection

La classe LocalConnection permet de développer des fichiers SWF qui peuvent échanger des instructions avec d'autres fichiers SWF sans utiliser la méthode `fscommand()` ni JavaScript. Les objets LocalConnection peuvent uniquement communiquer entre des fichiers SWF exécutés sur le même ordinateur client, mais ils peuvent concerner différentes applications. Par exemple, un fichier SWF exécuté dans un navigateur et un fichier SWF de projection peuvent partager des informations, le fichier de projection se chargeant de maintenir les informations locales et le fichier SWF du navigateur se connectant à distance. (Un fichier de projection est un fichier SWF enregistré dans un format tel qu'il peut s'exécuter de manière autonome ; il n'est donc pas nécessaire de disposer de Flash Player pour le lire, car il est incorporé dans le fichier exécutable.)

Les objets LocalConnection peuvent être utilisés pour communiquer entre des SWF utilisant différentes versions d'ActionScript :

- Les objets LocalConnection créés dans ActionScript 1.0 ou 2.0 peuvent communiquer avec les objets LocalConnection créés dans ActionScript 3.0.
- Les objets LocalConnection créés dans ActionScript 1.0 ou 2.0 peuvent communiquer avec les objets LocalConnection créés dans ActionScript 3.0.

Flash Player gère automatiquement les communications entre les objets LocalConnection de versions différentes.

La meilleure façon d'utiliser un objet LocalConnection est d'autoriser la communication uniquement entre des objets LocalConnection se trouvant dans le même domaine. Ainsi, vous évitez les problèmes de sécurité. Toutefois, si vous devez autoriser la communication entre les domaines, vous pouvez procéder de différentes façons pour implémenter vos mesures de sécurité. Pour plus d'informations, reportez-vous à la section consacrée au paramètre `connectionName` de la méthode `send()`, et aux entrées `allowDomain()` et `domain` dans la description de la classe LocalConnection du Guide de référence du langage et des composants ActionScript 3.0.

Il est possible d'utiliser des objets LocalConnection pour envoyer et recevoir des données au sein d'un fichier SWF unique. Toutefois, Adobe ne recommande pas cette pratique. Vous devriez plutôt utiliser des objets partagés.

Les méthodes de rappel peuvent être ajoutées aux objets LocalConnection de trois manières :

- Créer des sous-classes de LocalConnection et ajouter des méthodes
- Définir la propriété `LocalConnection.client` sur un objet qui implémente ces méthodes
- Créer une classe dynamique qui étend la classe LocalConnection et y joindre dynamiquement des méthodes

La première manière d'ajouter des méthodes de rappel est d'étendre la classe LocalConnection. Vous pouvez définir les méthodes au sein de la classe personnalisée, plutôt que de les ajouter dynamiquement à l'occurrence de LocalConnection. Ceci est illustré par le code suivant :.

```
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);
        }
    }
}
```

Pour créer une nouvelle occurrence de la classe DynamicLocalConnection, vous pouvez utiliser le code suivant :

```
var serverLC:CustomLocalConnection;
serverLC = new CustomLocalConnection("serverName");
```

La deuxième manière d'ajouter des méthodes de rappel consiste à utiliser la propriété `LocalConnection.client`. Il s'agit de créer une classe personnalisée et d'affecter une nouvelle occurrence à la propriété `client`, comme le montre le code suivant :

```
var lc:LocalConnection = new LocalConnection();
lc.client = new CustomClient();
```

La propriété `LocalConnection.client` indique les méthodes de rappel d'objet à appeler. Dans le code précédent la propriété `client` était définie sur une nouvelle occurrence de la classe personnalisée CustomClient. La valeur par défaut de la propriété `client` est l'occurrence de LocalConnection actuelle. Vous pouvez utiliser la propriété `client` si vous disposez de deux gestionnaires de données dotés du même jeu de méthodes mais qui agissent différemment ; par exemple, dans une application où un bouton situé dans une fenêtre permet d'afficher le contenu d'une deuxième fenêtre.

Pour créer la classe CustomClient, vous pouvez utiliser le code suivant :

```
package
{
    public class CustomClient extends Object
    {
        public function onMethod(timeString:String):void
        {
            trace("onMethod called at: " + timeString);
        }
    }
}
```

La troisième manière d'ajouter des méthodes de rappel consiste à créer une classe dynamique et d'y joindre dynamiquement des méthodes. Elle se rapproche de l'utilisation de la classe LocalConnection dans les versions précédentes d'ActionScript, comme le montre le code suivant :

```
import flash.net.LocalConnection;
dynamic class DynamicLocalConnection extends LocalConnection {}
```

Les méthodes de rappel peuvent être ajoutées dynamiquement à cette classe à l'aide du code suivant :

```
var connection:DynamicLocalConnection = new DynamicLocalConnection();
connection.onMethod = this.onMethod;
// Add your code here.
public function onMethod(timeString:String):void
{
    trace("onMethod called at: " + timeString);
}
```

Cette manière d'ajouter des méthodes de rappel est déconseillée, car le code n'est pas vraiment portable. En outre, cette méthode de création de connexions locales pourrait présenter des problèmes de performance car l'accès aux propriétés dynamiques prend bien plus de temps que l'accès aux propriétés scellées.

Envoi de messages entre deux occurrences de Flash Player

La classe LocalConnection permet de communiquer entre différentes occurrences de Flash Player et d'Adobe AIR. Par exemple, vous pouvez utiliser plusieurs occurrences de Flash Player sur une même page Web ou vous servir d'une occurrence de Flash Player pour récupérer des données dans une occurrence de Flash Player affichée dans une fenêtre distincte.

Le code suivant définit l'objet LocalConnection qui joue le rôle de serveur et accepte les appels entrants en provenance d'autres occurrences de Flash Player :

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

Ce code crée un objet LocalConnection nommé `lc` et définit la propriété `client` sur une classe personnalisée, CustomClient1. Lorsqu'une autre occurrence de Flash Player appelle une méthode dans cette occurrence de LocalConnection, Flash Player recherche la méthode dans la classe CustomClient1.

Dès qu'une occurrence de Flash Player se connecte à ce fichier SWF et essaie d'appeler l'une des méthodes de la connexion local, la requête est envoyée à la classe spécifiée par la propriété `client`, à savoir la classe 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");
        }
    }
}
```

Pour créer un serveur LocalConnection, appelez la méthode `LocalConnection.connect()` et fournissez un nom de connexion unique. Si une connexion est déjà ouverte avec le nom choisi, une erreur ArgumentError est générée, ce qui indique que la tentative de connexion a échoué parce que l'objet était déjà connecté.

L'extrait de code suivant illustre comment créer une nouvelle connexion de socket appelée `conn1` :

```
try
{
    connection.connect("conn1");
}
catch (error:ArgumentError)
{
    trace("Error! Server already exists\n");
}
```

La connexion au fichier SWF principal à partir d'un fichier SWF secondaire nécessite que vous créiez un objet LocalConnection dans l'objet LocalConnection d'envoi, puis que vous appeliez la méthode `LocalConnection.send()` avec le nom de la connexion et le nom de la méthode à exécuter. Par exemple, pour connecter un objet LocalConnection créé précédemment, vous pouvez utiliser le code suivant :

```
sendingConnection.send("conn1", "doQuit");
```

Ce code établit une connexion `conn1` à l'objet LocalConnection existant, puis appelle la méthode `doQuit()` dans le fichier SWF distant. Si vous souhaitez envoyer des paramètres au fichier SWF distant, spécifiez les arguments supplémentaires après le nom de la méthode `send()`, comme le montre l'extrait de code suivant :

```
sendingConnection.send("conn1", "doMessage", "Hello world");
```

Connexion à des documents SWF de domaines différents

Pour autoriser uniquement les communications issues de domaines précis, appelez la méthode `allowDomain()` ou `allowInsecureDomain()` de la classe LocalConnection et transmettez la liste des domaines autorisés à accéder à cet objet LocalConnection.

Dans les versions antérieures d'ActionScript, `LocalConnection.allowDomain()` et `LocalConnection.allowInsecureDomain()` étaient des méthodes de rappel que les développeurs devaient implémenter et qui devaient renvoyer une valeur booléenne. Dans ActionScript 3.0, `LocalConnection.allowDomain()` et `LocalConnection.allowInsecureDomain()` sont deux méthodes intégrées, que les développeurs peuvent appeler de la même façon que `Security.allowDomain()` et `Security.allowInsecureDomain()`, en transmettant un ou plusieurs noms de domaines à autoriser.

Deux valeurs spéciales peuvent être transmises aux méthodes `LocalConnection.allowDomain()` et `LocalConnection.allowInsecureDomain()` : `*` et `localhost`. L'astérisque `(*)` permet d'accéder à tous les domaines. La chaîne `localhost` permet d'appeler le fichier SWF à partir des fichiers SWF installés localement.

Dans Flash Player 8, des restrictions de sécurité relatives aux fichiers SWF locaux ont été introduites. Un fichier SWF autorisé à accéder à Internet n'a pas accès au système de fichiers local. Si vous spécifiez `localhost`, tout fichier SWF local peut accéder à ce fichier SWF. Si la méthode `LocalConnection.send()` tente de communiquer avec un fichier SWF à partir d'un sandbox de sécurité auquel le code d'appel n'a pas accès, un événement `securityError` (`SecurityErrorEvent.SECURITY_ERROR`) est distribué. Pour résoudre l'erreur, vous pouvez spécifier le domaine de l'appelant dans la méthode `LocalConnection.allowDomain()` de la cible.

Si vous implémentez la communication uniquement entre les fichiers SWF du même domaine, vous pouvez spécifier un paramètre `connectionName` qui ne commence pas par un trait de soulignement (_) et ne renvoie pas à un nom de domaine (par exemple `myDomain:connectionName`). Utilisez la même chaîne dans la commande `LocalConnection.connect(connectionName)`.

Si vous implémentez la communication entre les fichiers SWF de différents domaines, spécifiez un paramètre `connectionName` qui commence par un trait de soulignement. L'ajout d'un trait de soulignement améliore la portabilité entre domaines du fichier SWF contenant l'objet LocalConnection de réception. Les cas de figure possibles sont les suivants :

- Si la chaîne associée à `connectionName` ne commence pas par un trait de soulignement, Flash Player ajoute un préfixe au nom de superdomaine et deux-points (par exemple, `myDomain:connectionName`). Vous avez ainsi la garantie que votre connexion n'entrera pas en conflit avec les connexions de même nom dans d'autres domaines, mais tous les objets LocalConnection d'envoi doivent spécifier ce superdomaine (par exemple, `myDomain:connectionName`). Si le fichier SWF associé à l'objet LocalConnection de réception est déplacé vers un autre domaine, le lecteur modifie le préfixe afin qu'il reflète le nouveau superdomaine (par exemple, `anotherDomain:connectionName`). Tous les objets LocalConnection d'envoi doivent être modifiés manuellement pour pointer vers le nouveau superdomaine.
- Si la chaîne associée à `connectionName` commence par un trait de soulignement (par exemple, `_connectionName`), Flash Player ne lui ajoute pas de préfixe. Cela signifie que les objets LocalConnection de réception et d'envoi utilisent des chaînes identiques pour `connectionName`. Si l'objet de réception utilise `LocalConnection.allowDomain()` pour spécifier que les connexions seront acceptées à partir de tous les domaines, le fichier SWF contenant l'objet LocalConnection de réception peut être déplacé vers un autre domaine, sans qu'il soit nécessaire de modifier les objets LocalConnection d'envoi.

Connexions socket

Il existe deux types de connexion socket différents dans ActionScript 3.0 : les connexions socket XML et les connexions socket binaires. Un socket XML vous permet de vous connecter à un serveur distant et de créer une connexion serveur qui reste ouverte jusqu'à sa fermeture explicite. Vous pouvez ainsi échanger des données XML entre un serveur et un client sans avoir à ouvrir à chaque fois une nouvelle connexion serveur. Autre avantage de l'utilisation d'un serveur socket XML : l'utilisateur n'a pas besoin de demander explicitement les données. Vous pouvez envoyer des données du serveur sans requête préalable, et ce à chaque client connecté au serveur socket XML.

Les connexions de socket XML nécessitent la présence d'un fichier de régulation des sockets sur le serveur cible. Pour plus d'informations, consultez les sections « Contrôles de site Web (fichiers de régulation) » à la page 721 et « Connexion aux sockets » à la page 736.

Une connexion socket binaire est semblable à un socket XML à la différence que le client et le serveur n'ont pas besoin d'échanger obligatoirement des paquets XML. La connexion peut en effet transférer des données au format binaire. Vous pouvez ainsi vous connecter à une large gamme de services, notamment des serveurs de messagerie (POP3, SMTP et IMAP) et d'informations (NNTP).

La classe Socket

Nouveauté d'ActionScript 3.0, la classe Socket permet au code d'établir des connexions socket et de lire et d'écrire des données binaires brutes. Elle est semblable à la classe XMLSocket mais n'impose pas de contrainte quant au format des données reçues ou transmises. La classe Socket est utile si vous utilisez des serveurs faisant appel à des protocoles binaires. Grâce aux connexions socket binaires, vous pouvez écrire du code permettant l'interaction entre différents protocoles Internet, par exemple POP3, SMTP, IMAP et NNTP. En échange, cela permet à Flash Player de se connecter à des serveurs de messagerie et d'informations.

Flash Player peut communiquer directement avec un serveur en utilisant directement le protocole binaire de ce serveur. Certains serveurs utilisent l'ordre d'octets « gros-boutiste », d'autres l'ordre « petit-boutiste ». La plupart des serveurs sur Internet utilise l'ordre gros-boutiste car il s'agit de l'ordre d'octets du réseau. L'ordre petit-boutiste s'est répandu en raison de son utilisation par l'architecture Intel® x86. Vous devez utiliser l'ordre d'octets correspondant au serveur qui envoie et reçoit les données. Toutes les opérations sont effectuées par les interfaces IDataInput et IDataOutput, et les classes qui les implémentent (ByteArray, Socket et URLStream) sont par défaut encodées au format gros-boutiste (l'octet le plus significatif en premier). On vise ainsi à respecter Java et l'ordre d'octets officiel des réseaux. Pour modifier l'ordre d'octets à utiliser, vous pouvez définir la propriété `endian` sur `Endian.BIG_ENDIAN` ou `Endian.LITTLE_ENDIAN`.

La classe Socket hérite de toutes les méthodes implémentées par les interfaces IDataInput et IDataOutput (dans le package flash.utils). Ces méthodes sont à utiliser pour l'écriture et la lecture de la classe Socket.

La classe XMLSocket

ActionScript fournit une classe XMLSocket intégrée qui vous permet d'établir une connexion continue avec un serveur. Cette connexion ouverte supprime les périodes d'attente et sert souvent dans des applications en temps réel telles que les dialogues en ligne ou les jeux multijoueurs. Une solution de dialogue en ligne par HTTP classique interroge fréquemment le serveur et télécharge les nouveaux messages à l'aide d'une requête HTTP. Par contraste, une solution de dialogue en ligne XMLSocket maintient une connexion ouverte avec le serveur, permettant à celui-ci d'envoyer immédiatement les messages entrants sans requête du client.

Pour créer une connexion socket, vous devez créer une application côté serveur qui attendra la requête de connexion socket et enverra une réponse au fichier SWF. Ce type d'application côté serveur peut être écrit dans un langage tel que Java, Python ou Perl. Pour utiliser la classe XMLSocket, l'ordinateur serveur doit exécuter un démon capable de lire le protocole utilisé par la classe XMLSocket. Le protocole est décrit dans la liste suivante :

- Les messages XML sont envoyés via une connexion socket à flux TCP/IP bidirectionnel simultané.
- Chaque message XML est un document XML complet, terminé par un octet nul (0).
- Un nombre illimité de messages XML peut être envoyé et reçu via une connexion XMLSocket.

La classe XMLSocket ne peut pas automatiquement emprunter un tunnel à travers les pare-feux car, contrairement au protocole RTMP (Real-Time Messaging Protocol), le XMLSocket n'a pas de capacité de tunneling HTTP. Si vous devez utiliser le tunneling HTTP, envisagez l'emploi de Flash Remoting ou Flash Media Server (qui prend en charge RTMP).

Remarque : *la configuration d'un serveur en vue de la communication avec un objet XMLSocket peut être difficile à réaliser. Si votre application ne nécessite pas d'interactivité en temps réel, utilisez la classe URLLoader, plutôt que la classe XMLSocket.*

Vous pouvez utiliser les méthodes `XMLSocket.connect()` et `XMLSocket.send()` de la classe XMLSocket pour transférer un objet XML vers et à partir d'un serveur sur une connexion socket. La méthode `XMLSocket.connect()` établit une connexion socket avec le port d'un serveur Web. La méthode `XMLSocket.send()` transmet un objet XML au serveur spécifié dans la connexion socket.

Lorsque vous invoquez la méthode `XMLSocket.connect()`, Flash Player ouvre une connexion TCP/IP avec le serveur et garde cette connexion ouverte jusqu'à ce que l'un des événements suivants se produise :

- La méthode `XMLSocket.close()` de la classe XMLSocket est appelée.
- Il n'existe plus aucune référence à l'objet XMLSocket.
- Flash Player se ferme.
- La connexion est interrompue (le modem est déconnecté, par exemple).

Création et connexion à un serveur socket XML Java

Le code suivant illustre un simple serveur XMLSocket écrit en langage Java qui accepte les connexion entrantes et affichent les messages reçus dans la fenêtre d'invite de commande. Par défaut, un nouveau serveur est créé sur le port 8080 de votre machine locale, mais vous pouvez spécifier un numéro de port différent en lançant le serveur à partir de la ligne de commande.

Créez un document texte et ajoutez-y le code suivant :

```
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);
    }
}
```

Enregistrez le document sur le disque dur sous le nom SimpleServer.java et compilez-le à l'aide d'un compilateur Java, qui crée un fichier de classe Java nommé SimpleServer.class.

Vous pouvez lancer le serveur XMLSocket via la ligne de commande en tapant `java SimpleServer`. Le fichier SimpleServer.class peut se situer à tout emplacement sur l'ordinateur local ou le réseau ; il n'est pas nécessaire de le placer dans le répertoire racine du serveur Web.

Si vous ne pouvez pas lancer le serveur parce que des fichiers ne se trouvent pas dans le chemin de classe Java, essayez de le faire avec `java -classpath. SimpleServer`.

Pour établir une connexion au serveur XMLSocket à partir de votre application ActionScript, vous devez créer une nouvelle occurrence de la classe XMLSocket et appeler la méthode `XMLSocket.connect()` tout en transférant le nom d'hôte et le numéro de port, comme suit :

```
var xmlsock:XMLSocket = new XMLSocket();
xmlsock.connect("127.0.0.1", 8080);
```

Dès que vous recevez des données du serveur, l'événement `data` (`flash.events.DataEvent.DATA`) est distribué :

```
xmlsock.addEventListener(DataEvent.DATA, onData);
private function onData(event:DataEvent):void
{
    trace("[" + event.type + "] " + event.data);
}
```

Pour envoyer des données au serveur XMLSocket, utilisez la méthode `XMLSocket.send()` et transmettez un objet XML ou une chaîne. Flash Player convertit le paramètre fourni en objet String et envoie au serveur XMLSocket son contenu suivi d'un octet nul (0) :

```
xmlsock.send(xmlFormattedData);
```

La méthode `XMLSocket.send()` ne renvoie pas de valeur indiquant si les données ont bien été transmises. Si une erreur se produit pendant la tentative d'envoi des données, une erreur IOError est renvoyée.

Chaque message que vous envoyez au serveur socket doit se terminer par un caractère de changement de ligne (`\n`).

Stockage des données locales

Un objet partagé, parfois appelé cookie Flash, est un fichier de données qui peut être créé sur votre ordinateur par les sites que vous visitez. Les objets partagés servent le plus souvent à améliorer votre navigation sur le Web, par exemple en vous permettant de personnaliser l'aspect d'un site Web que vous consultez fréquemment. Pris isolément, les objets partagés ne peuvent interagir avec les données de votre ordinateur. Point important : les objets partagés ne peuvent en aucun cas accéder à votre adresse électronique (ou autre information personnelle) ni la conserver sans votre consentement formel.

Il est possible de créer des occurrences d'objets partagés à l'aide des méthodes statiques `SharedObject.getLocal()` ou `SharedObject.getRemote()`. La méthode `getLocal()` essaie de charger un objet partagé persistant localement, disponible uniquement sur le client actuel. A l'inverse, la méthode `getRemote()` tente de charger un objet partagé distant, susceptible d'être partagé par plusieurs clients au moyen d'un serveur tel que Flash Media Server. S'il n'existe aucun objet partagé local ou distant, les méthodes `getLocal()` et `getRemote()` créent une nouvelle occurrence de SharedObject.

Le code suivant essaie de charger un objet partagé local nommé `test`. Si cet objet n'existe pas, un objet partagé est créé avec le même nom.

```
var so:SharedObject = SharedObject.getLocal("test");
trace("SharedObject is " + so.size + " bytes");
```

Si aucun objet partagé nommé test n'est trouvé, un objet est créé, d'une taille de 0 octet. Si l'objet partagé existait précédemment, sa taille actuelle (en octets) est rétablie.

Vous pouvez stocker des données dans un objet partagé en attribuant des valeurs à l'objet de données, comme le montre l'exemple suivant :

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.now = new Date().time;
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

S'il existe déjà un objet partagé nommé `test` avec le paramètre `now`, la valeur existante est remplacée. La propriété `SharedObject.size` vous permet de déterminer si un objet partagé existe déjà, comme illustré ci-après :

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
    // Shared object doesn't exist.
    trace("created...");
    so.data.now = new Date().time;
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

Le code précédent utilise le paramètre `size` afin de déterminer si l'occurrence d'objet partagé du nom spécifié existe déjà. Si vous testez le code suivant, vous remarquerez qu'à chaque exécution, l'objet partagé est recréé. Pour enregistrer un objet partagé sur le disque dur de l'utilisateur, vous devez explicitement appeler la méthode `SharedObject.flush()`, comme dans l'exemple ci-après :

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
    // Shared object doesn't exist.
    trace("created...");
    so.data.now = new Date().time;
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

Si vous utilisez la méthode `flush()` pour écrire des objets partagés sur le disque dur de l'ordinateur, vous devez vérifier avec soin si l'utilisateur a explicitement désactivé le stockage local à l'aide du Gestionnaire de paramètres de Flash Player (www.macromedia.com/support/documentation/fr/flashplayer/help/settings_manager07.html), comme illustré dans cet exemple :

```
var so:SharedObject = SharedObject.getLocal("test");
trace("Current SharedObject size is " + so.size + " bytes.");
so.flush();
```

Il est possible de récupérer des valeurs dans un objet partagé en spécifiant le nom de la propriété dans la propriété `data` de l'objet partagé. Par exemple, si vous exécutez le code suivant, Flash Player affiche depuis combien de minutes l'occurrence de SharedObject a été créée :

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
    // Shared object doesn't exist.
    trace("created...");
    so.data.now = new Date().time;
}
var ageMS:Number = new Date().time - so.data.now;
trace("SharedObject was created " + Number(ageMS / 1000 / 60).toPrecision(2) + " minutes ago");
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

A la première exécution du code précédent, une nouvelle occurrence de SharedObject appelée `test` est créée, avec une taille initiale de 0 octet. Parce que la taille initiale est nulle, l'instruction `if` renvoie la valeur `true` et une nouvelle propriété `now` est ajoutée à l'objet partagé local. L'ancienneté de l'objet partagé est calculée par soustraction de la valeur de la propriété `now` de l'heure actuelle. A chaque exécution suivante du code ci-dessus, la taille de l'objet partagé doit être supérieure à zéro ; le code peut effectuer un suivi du nombre de minutes écoulées depuis la création de l'objet partagé.

Affichage du contenu d'un objet partagé

Des valeurs sont stockées dans un objet partagé, au sein de la propriété `data`. Vous pouvez passer en boucle chaque valeur d'une occurrence d'objet partagé à l'aide de la boucle `for..in`, comme le montre l'exemple suivant :

```
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]);
}
```

Création d'un SharedObject sécurisé

Lorsque vous créez un SharedObject local ou distant à l'aide de `getLocal()` ou `getRemote()`, un paramètre facultatif nommé `secure` détermine si l'accès à l'objet partagé se limite aux fichiers SWF diffusés via une connexion HTTPS. Si ce paramètre a la valeur `true` et que votre fichier SWF est diffusé sur HTTPS, Flash Player crée un nouvel objet sécurisé ou obtient une référence à un objet partagé sécurisé existant. Cet objet partagé sécurisé peut uniquement être lu ou écrit par des fichiers SWF reçus via des connexions HTTPS appelant `SharedObject.getLocal()` avec le paramètre secure réglé sur `true`. Si ce paramètre a la valeur `false` et que votre fichier SWF est diffusé sur HTTPS, Flash Player crée un nouvel objet partagé ou obtient une référence à un objet partagé existant.

Cet objet partagé peut être lu ou écrit par des fichiers SWF reçus via des connexions autres que HTTPS. Si votre fichier SWF est diffusé via une connexion autre que HTTPS et que vous essayez de régler ce paramètre sur `true`, la création d'un objet partagé (ou l'accès à un objet partagé sécurisé précédemment créé) échoue, une erreur est renvoyée et l'objet partagé devient `null`. Si vous tentez d'exécuter l'extrait de code suivant à partir d'un connexion non HTTPS, la méthode `SharedObject.getLocal()` renvoie une erreur :

```
try
{
    var so:SharedObject = SharedObject.getLocal("contactManager", null, true);
}
catch (error:Error)
{
    trace("Unable to create SharedObject.");
}
```

Quelle que soit la valeur de ce paramètre, les objets partagés créés sont comptabilisés dans la quantité d'espace disque total autorisée pour un domaine.

Utilisation des fichiers de données

Un objet FileReference représente un fichier de données stocké sur un client ou un serveur. Les méthodes de la classe FileReference permettent à votre application de charger et d'enregistrer localement des fichiers de données et de transférer ces derniers entre la machine locale et un serveur distant.

La classe FileReference propose deux approches distinctes pour charger, transférer et enregistrer les fichiers de données. Depuis qu'elle a été introduite, la classe FileReference inclut la méthode `browse()`, qui permet à l'utilisateur de sélectionner un fichier, la méthode `upload()`, qui permet de transférer les données du fichier à un serveur distant, et la méthode `download()`, permettant d'extraire ces données du serveur en vue de les enregistrer dans un fichier local. A partir de Flash Player 10 et Adobe AIR 1.5, la classe FileReference dispose des méthodes `load()` et `save()` qui vous permettent d'accéder aux fichiers locaux et de les enregistrer directement. L'utilisation de ces méthodes est similaire aux méthodes du même nom dont disposent les classes URLLoader et Loader. Cette section aborde les deux utilisations de la classe FileReference.

Remarque : *le moteur d'exécution AIR fournit d'autres classes (intégrées au package flash.filesystem) pour la manipulation des fichiers et du système de fichiers local. Les classes flash.filesystem proposent davantage de fonctionnalités que la classe FileReference, mais elles ne sont prises en charge que par AIR et non par Flash Player.*

Classe FileReference

Chaque objet FileReference se réfère à un fichier de données local hébergé sur la machine locale. Les propriétés de la classe FileReference contiennent des informations sur la taille, le type, le nom, l'extension, le créateur, la date de création et la date de modification du fichier.

Remarque : *la propriété `creator` est prise en charge sous Mac OS uniquement. Toutes les autres plates-formes renvoient la valeur `null`.*

Remarque : *la propriété `extension` n'est prise en charge que par le moteur d'exécution AIR.*

Pour créer une occurrence de la classe FileReference, procédez comme suit, au choix :

- Utilisez l'opérateur `new`, comme indiqué dans le code suivant :

```
import flash.net.FileReference;
var fileRef:FileReference = new FileReference();
```

- Appelez la méthode `FileReferenceList.browse()`, qui ouvre une boîte de dialogue et invite l'utilisateur à sélectionner un ou plusieurs fichiers à télécharger. Elle crée ensuite un tableau d'objets FileReference si l'utilisateur réussit à sélectionner un ou plusieurs fichiers.

Une fois l'objet FileReference créé, vous pouvez procéder comme suit :

- Appelez la méthode `FileReference.browse()`, qui ouvre une boîte de dialogue et invite l'utilisateur à sélectionner un fichier unique dans le système de fichiers local. Cette opération est généralement effectuée avant un nouvel appel de la méthode `FileReference.upload()` pour télécharger le fichier sur un serveur distant ou un appel de la méthode `FileReference.load()` pour ouvrir un fichier local.
- Appelez la méthode `FileReference.download()`. Elle ouvre une boîte de dialogue qui permet à l'utilisateur de sélectionner l'emplacement d'enregistrement d'un nouveau fichier. Elle télécharge ensuite les données du serveur et les stocke dans le nouveau fichier.
- Appelez la méthode `FileReference.load()`. Cette méthode commence le chargement de données à partir d'un fichier précédemment sélectionné à l'aide de la méthode `browse()`. Il est impossible d'appeler la méthode `load()` tant que l'opération `browse()` n'est pas terminée (c'est-à-dire lorsque l'utilisateur sélectionne un fichier).
- Appelez la méthode `FileReference.save()`. Cette méthode ouvre une boîte de dialogue et invite l'utilisateur à sélectionner un emplacement de fichier unique sur le système de fichiers local. Elle permet ensuite d'enregistrer les données à l'emplacement spécifié.

Remarque : *vous ne pouvez exécuter qu'une seule méthode `browse()`, `download()` ou `save()` à la fois, car une seule boîte de dialogue peut être ouverte à un moment donné.*

Les propriétés de l'objet FileReference, telles que `name`, `size` ou `modificationDate`, ne sont pas renseignées tant que l'un des événements suivants ne s'est pas produit :

- La méthode `FileReference.browse()` ou `FileReferenceList.browse()` a été appelée et l'utilisateur a sélectionné un fichier dans la boîte de dialogue.
- La méthode `FileReference.download()` a été appelée et l'utilisateur a stipulé un nouvel emplacement de fichier par le biais de la boîte de dialogue.

Remarque : *lors d'un téléchargement, seule la propriété `FileReference.name` est renseignée avant la fin du téléchargement. Une fois que le fichier est téléchargé, toutes les propriétés sont disponibles.*

Lors de l'exécution des appels de la méthode `FileReference.browse()`, `FileReferenceList.browse()`, `FileReference.download()`, `FileReference.load()` ou `FileReference.save()`, la plupart des lecteurs poursuivent la lecture du fichier SWF, ainsi que la distribution d'événements et l'exécution du code.

Pour les opérations de chargement ou téléchargement, un fichier SWF peut uniquement accéder aux fichiers de son propre domaine, ce qui comprend tous les domaines spécifiés par un fichier de régulation. Vous devez placer un fichier de régulation sur le serveur contenant le fichier, si ce serveur ne se trouve pas sur le même domaine que le fichier SWF ayant initié le chargement ou le téléchargement.

Chargement de données à partir d'un fichier

La méthode `FileReference.load()` vous permet de charger des données en mémoire à partir d'un fichier local. Votre code doit tout d'abord appeler la méthode `FileReference.browse()` pour que l'utilisateur puisse sélectionner le fichier à charger.

La méthode `FileReference.load()` renvoie une valeur immédiatement après avoir été appelée, mais les données en cours de chargement ne sont pas disponibles tout de suite. L'objet FileReference distribue des événements pour appeler les méthodes d'écouteur à chaque étape du processus de chargement.

L'objet FileReference distribue les événements suivants pendant le processus de chargement.

- Evénement `open` (`Event.OPEN`) : distribué lorsque l'opération de chargement commence.
- Evénement `progress` (`ProgressEvent.PROGRESS`) : distribué régulièrement lorsque le fichier lit des octets de données.
- Evénement `complete` (`Event.COMPLETE`) : distribué en cas de réussite de l'opération de chargement.
- Evénement `ioError` (`IOErrorEvent.IO_ERROR`) : distribué si le processus de chargement échoue en raison d'une erreur d'entrée/sortie lors de l'ouverture ou de la lecture des données du fichier.

Une fois que l'objet FileReference distribue l'événement complete, il est possible d'accéder aux données chargées comme un élément ByteArray dans la propriété `data` de l'objet FileReference.

L'exemple suivant indique comment inviter l'utilisateur à sélectionner un fichier, puis à charger les données de ce dernier en mémoire :

```
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.");
        }
    }
}
```

Le code d'exemple crée tout d'abord l'objet FileReference nommé `fileRef`, puis appelle sa méthode `browse()`. Une boîte de dialogue s'ouvre et l'utilisateur est invité à sélectionner un fichier. Une fois le fichier sélectionné, la méthode `onFileSelected()` est appelée. Cette méthode ajoute des écouteurs aux événements `progress` et `complete`, puis appelle la méthode `load()` de l'objet FileReference. Les autres méthodes de gestionnaire de cet exemple se contentent de générer des messages qui indiquent le déroulement de l'opération de chargement. Une fois le chargement terminé, l'application affiche le contenu du fichier chargé à l'aide de la méthode `trace()`.

Enregistrement de données dans des fichiers locaux

La méthode `FileReference.save()` vous permet d'enregistrer des données dans un fichier local. Elle commence par ouvrir une boîte de dialogue qui permet à l'utilisateur d'entrer un nouveau nom de fichier et l'emplacement d'enregistrement du fichier. Une fois le nom de fichier et l'emplacement sélectionnés, les données sont écrites dans le nouveau fichier. Lorsque le fichier est enregistré, les propriétés de l'objet FileReference sont renseignées à partir des propriétés du fichier local.

Remarque : *votre code ne doit appeler la méthode `FileReference.save()` qu'en réponse à un événement utilisateur, tel qu'un gestionnaire associé à un événement de type clic de souris ou pression de touche. Dans le cas contraire, une erreur est renvoyée.*

La méthode `FileReference.save()` est renvoyée juste après son appel. L'objet FileReference distribue ensuite des événements pour appeler les méthodes d'écouteur à chaque étape du processus d'enregistrement de fichier.

L'objet FileReference distribue les événements suivants au cours du processus d'enregistrement de fichier :

- Evénement `select` (`Event.SELECT`) : distribué lorsque l'utilisateur indique l'emplacement et le nom du nouveau fichier à enregistrer.
- Evénement `cancel` (`Event.CANCEL`) : distribué lorsque l'utilisateur clique sur le bouton Annuler dans la boîte de dialogue.

- Evénement `open` (`Event.OPEN`) : distribué lorsque l'opération d'enregistrement commence.
- Evénement `progress` (`ProgressEvent.PROGRESS`) : distribué régulièrement pendant l'enregistrement des octets de données dans le fichier.
- Evénement `complete` (`Event.COMPLETE`) : distribué en cas de réussite de l'opération d'enregistrement.
- Evénement `ioError` (`IOErrorEvent.IO_ERROR`) : distribué si le processus d'enregistrement échoue en raison d'une erreur d'entrée/sortie lors d'une tentative d'enregistrement des données dans le fichier.

Le type d'objet transmis dans le paramètre `data` de la méthode `FileReference.save()` détermine le mode d'écriture des données dans le fichier :

- S'il s'agit d'une valeur String, les données sont enregistrées en tant que fichier texte à l'aide de l'encodage UTF-8.
- S'il s'agit d'un objet XML, elles sont écrites dans un fichier XML en conservant l'ensemble du formatage.
- S'il s'agit d'un objet ByteArray, leur contenu est écrit directement dans le fichier sans conversion.
- S'il s'agit d'un autre type d'objet, la méthode `FileReference.save()` appelle la méthode `toString()` de l'objet, puis enregistre la valeur String résultante dans un fichier texte UTF-8. S'il est impossible d'appeler la méthode `toString()` de l'objet, une erreur est renvoyée.

Si la valeur du paramètre `data` est `null`, une erreur est renvoyée.

Le code suivant étend l'exemple précédent pour la méthode `FileReference.load()`. Une fois les données lues dans le fichier, cet exemple invite l'utilisateur à entrer un nom de fichier, puis enregistre les données dans un nouveau fichier :

```
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.");
        }
    }
}
```

Lorsque toutes les données du fichier ont été chargées, la méthode `onComplete()` est appelée. La méthode `onComplete()` supprime les écouteurs associés aux événements de chargement, puis appelle la méthode `saveFile()`. La méthode `saveFile()` appelle la méthode `FileReference.save()`, qui ouvre une nouvelle boîte de dialogue dans laquelle l'utilisateur entre un nouveau nom de fichier et l'emplacement d'enregistrement de ce dernier. Les autres méthodes d'écouteur d'événement tracent le déroulement du processus d'enregistrement du fichier jusqu'à ce qu'il soit terminé.

Chargement de fichiers sur un serveur

Pour charger des fichiers sur un serveur, commencez par appeler la méthode `browse()` pour permettre à l'utilisateur de sélectionner un ou plusieurs fichiers. Après l'appel de la méthode `FileReference.upload()`, le fichier sélectionné est transféré sur le serveur. Si l'utilisateur sélectionne plusieurs fichiers à l'aide de la méthode `FileReferenceList.browse()`, Flash Player crée un tableau de fichiers sélectionnés appelé `FileReferenceList.fileList`. Vous pouvez alors utiliser la méthode `FileReference.upload()` pour charger chaque fichier individuellement.

Remarque : *l'utilisation de la méthode `FileReference.browse()` ne vous permet de charger qu'un seul fichier à la fois. Pour que l'utilisateur puisse charger plusieurs fichiers, vous devez utiliser la méthode `FileReferenceList.browse()`.*

Par défaut, la boîte de dialogue de sélection de fichiers du système d'exploitation permet à l'utilisateur de choisir tout type de fichier sur l'ordinateur local. Les développeurs peuvent toutefois filtrer les types de fichier à l'aide de la classe FileFilter, en transmettant un tableau d'occurrences de filtres de fichiers à la méthode `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);
```

Une fois que l'utilisateur a sélectionné les fichiers et cliqué sur le bouton Ouvrir du sélecteur de fichier du système, l'événement `Event.SELECT` est distribué. Si vous utilisez la méthode `FileReference.browse()` pour sélectionner le fichier à charger, vous devez utiliser le code suivant pour envoyer le fichier au serveur Web :

```
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");
}
```

Avec la méthode `FileReference.upload()`, vous pouvez envoyer des données au serveur en utilisant les propriétés `URLRequest.method` et `URLRequest.data` en vue d'envoyer des variables à l'aide de la méthode `POST` ou `GET`.

Si vous tentez de charger un fichier à l'aide de la méthode `FileReference.upload()`, il est possible que les événements suivants soient distribués :

- Evénement `open` (`Event.OPEN`) : distribué lorsque l'opération de chargement commence.
- Evénement `progress` (`ProgressEvent.PROGRESS`) : distribué régulièrement pendant le chargement des octets de données du fichier.
- Evénement `complete` (`Event.COMPLETE`) : distribué en cas de réussite de l'opération de chargement.
- Evénement `httpStatus` (`HTTPStatusEvent.HTTP_STATUS`) : distribué lorsque le processus de chargement échoue en raison d'une erreur HTTP.
- Evénement `httpResponseStatus` (`HTTPStatusEvent.HTTP_RESPONSE_STATUS`) : distribué si un appel de la méthode `upload()` ou `uploadUnencoded()` tente d'accéder aux données via HTTP, et si Adobe AIR est capable de détecter et de renvoyer le code d'état de la requête.
- Evénement `securityError` (`SecurityErrorEvent.SECURITY_ERROR`) : distribué lorsqu'une opération de chargement échoue en raison d'une violation de la sécurité.
- Evénement `uploadCompleteData` (`DataEvent.UPLOAD_COMPLETE_DATA`) : distribué après réception des données par le serveur suite à un chargement réussi.
- Evénement `ioError` (`IOErrorEvent.IO_ERROR`) : distribué si le processus de chargement échoue pour l'une des raisons suivantes :
    - Une erreur d'entrée/sortie se produit dans Flash Player pendant la lecture, l'écriture ou la transmission du fichier.

- Le fichier SWF tente de charger un fichier sur un serveur nécessitant une authentification (un nom d'utilisateur et un mot de passe, par exemple). Au cours du chargement, Flash Player ne permet pas aux utilisateurs d'entrer des mots de passe.
- Le paramètre `url` contient un protocole incorrect. La méthode `FileReference.upload()` doit utiliser HTTP ou HTTPS.

Flash Player n'offre pas une prise en charge complète des serveurs nécessitant une authentification. Seuls les fichiers SWF s'exécutant dans un navigateur, via le module externe du navigateur ou le contrôle Microsoft ActiveX®, peuvent fournir une boîte de dialogue pour inviter l'utilisateur à entrer un nom et un mot de passe d'authentification, et ce uniquement pour les téléchargements. Le transfert de fichiers échoue si le chargement est effectué à l'aide du module externe ou du contrôle ActiveX, ou si un chargement/téléchargement est effectué par le biais du lecteur autonome ou externe.

Pour créer un script serveur dans ColdFusion de manière à accepter les chargements de fichier en provenance de Flash Player, vous pouvez utiliser un code semblable à celui-ci :

```
<cffile action="upload" filefield="Filedata" destination="#ExpandPath('./')#"
nameconflict="OVERWRITE" />
```

Ce code ColdFusion charge le fichier envoyé par Flash Player et l'enregistre dans le même répertoire que le modèle ColdFusion, en écrasant tout fichier existant du même nom. Cet exemple présente le minimum de code nécessaire à l'acceptation du chargement d'un fichier ; ce script ne doit pas être utilisé dans un environnement de production. Dans l'idéal, il faudrait ajouter un mécanisme de validation des données pour garantir que les utilisateurs chargent uniquement des types de fichiers autorisés, par exemple une image plutôt qu'un script côté serveur potentiellement dangereux.

Le code ci-après présente le chargement de fichiers via PHP, avec validation des données. Le script limite le nombre de fichiers chargés dans le répertoire cible à 10, vérifie que le fichier fait moins de 200 Ko et autorise uniquement le chargement et l'enregistrement de fichiers JPEG, GIF ou PNG.

```
<?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;
    }
}
?>
```

Vous pouvez transmettre des variables supplémentaires au script de chargement à l'aide de la méthode de requête `POST` ou `GET`. Pour envoyer des variables `POST` au script de chargement, vous pouvez utiliser le code suivant :

```
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");
}
```

L'exemple précédent crée un objet URLVariables à transmettre au script côté serveur distant. Dans les versions précédentes d'ActionScript, il était possible de transmettre des variables au script de chargement en passant des valeurs dans la chaîne de requête. ActionScript 3.0 vous permet de transmettre des variables au script distant à l'aide de l'objet URLRequest. Vous pouvez ainsi transmettre des données à l'aide de la méthode `POST` ou `GET`, ce qui simplifie et rationalise le transfert de gros volumes de données. Pour spécifier si les variables sont transmises à l'aide de la méthode de requête `GET` ou `POST`, il est possible de définir la propriété `URLRequest.method` sur `URLRequestMethod.GET` ou `URLRequestMethod.POST`, respectivement.

ActionScript 3.0 vous permet de remplacer le nom de champ par défaut du fichier à charger (`Filedata`) en ajoutant un deuxième paramètre à la méthode `upload()`, comme illustré dans l'exemple précédent (dans lequel la valeur par défaut `Filedata` est remplacée par `Custom1`).

Par défaut, Flash Player n'essaie pas d'effectuer un chargement de test ; vous pouvez toutefois le faire en transmettant la valeur `true` comme troisième paramètre de la méthode `upload()`. L'objectif du test est de vérifier que le chargement véritable se fera sans problème et que l'authentification du serveur, si nécessaire, réussira.

Remarque : *actuellement, le test du chargement s'effectue uniquement dans les versions Windows de Flash Player.*

Le script serveur qui gère le chargement doit attendre une requête HTTP `POST` comportant les éléments suivants :

- `Content-Type` avec la valeur `multipart/form-data`.
- `Content-Disposition` avec comme attribut `name` « `Filedata` » et comme attribut `filename` le nom du fichier d'origine. Pour spécifier un attribut `name`, transmettez une valeur pour le paramètre `uploadDataFieldName` dans la méthode `FileReference.upload()`.
- Le contenu binaire du fichier.

Voici un exemple de requête HTTP `POST` :

```
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,,,)
```

L'exemple de requête HTTP `POST` suivant envoie trois variables `POST` : `api_sig`, `api_key` et `auth_token`, puis utilise la valeur de champ de données `"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"

XXXXXXXXXXXXXXXXXXXXXXX
------------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--
```

Téléchargement de fichiers à partir d'un serveur

Vous pouvez autoriser les utilisateurs à télécharger des fichiers à partir d'un serveur grâce à la méthode `FileReference.download()`, qui prend deux paramètres : `request` et `defaultFileName`. Le premier paramètre est l'objet URLRequest contenant l'URL du fichier à télécharger. Le second est facultatif ; il permet de spécifier un nom de fichier par défaut qui apparaîtra dans la boîte de dialogue de téléchargement. Si vous ignorez le second paramètre, `defaultFileName`, le nom de fichier utilisé est dérivé de l'URL.

Le code suivant télécharge un fichier nommé index.xml à partir du même répertoire que le document SWF :

```
var request:URLRequest = new URLRequest("index.xml");
var fileRef:FileReference = new FileReference();
fileRef.download(request);
```

Pour utiliser comme nom par défaut currentnews.xml au lieu de index.xml, spécifiez le paramètre `defaultFileName`, comme le montre l'extrait de code suivant :

```
var request:URLRequest = new URLRequest("index.xml");
var fileToDownload:FileReference = new FileReference();
fileToDownload.download(request, "currentnews.xml");
```

Le changement de nom du fichier peut s'avérer très utile si le nom du fichier sur le serveur est peu évocateur ou généré automatiquement. Il est également judicieux de spécifier le paramètre `defaultFileName` lorsque vous téléchargez un fichier à l'aide d'un script côté serveur, au lieu d'effectuer un téléchargement direct. Par exemple, il est nécessaire de spécifier le paramètre `defaultFileName` si vous utilisez un script côté serveur qui télécharge des fichiers en fonction des variables URL qui lui sont transmises. Autrement, le nom par défaut du fichier téléchargé est le nom du script côté serveur.

Vous pouvez également envoyer des données au serveur avec l'appel de la méthode `download()` en ajoutant des paramètres à l'URL pour que le script serveur les analyse. L'extrait de code ActionScript 3.0 ci-après télécharge un document en fonction des paramètres transmis à un script ColdFusion :

```
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.");
            }
        }
    }
}
```

Le code suivant présente le script ColdFusion download.cfm, qui télécharge l'un des deux fichiers stockés sur le serveur en fonction de la valeur d'une variable URL :

```
<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>
```

La classe FileReferenceList

La classe FileReferenceList permet à l'utilisateur de sélectionner un ou plusieurs fichiers à charger dans un script côté serveur. Le chargement de fichiers est géré par la méthode `FileReference.upload()`, qui doit être appelée pour chaque fichier sélectionné.

Le code suivant crée deux objets FileFilter (`imageFilter` et `textFilter`) et les transmet sous forme de tableau à la méthode `FileReferenceList.browse()`. Ainsi, la boîte de dialogue du système d'exploitation propose deux types de fichiers possibles.

```
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.");
}
```

L'utilisation de la classe FileReferenceList pour autoriser le chargement de fichiers est semblable à l'utilisation de `FileReference.browse()`, à la différence que FileReferenceList permet de sélectionner plusieurs fichiers. En cas de sélection de fichiers multiples, il est nécessaire de charger chacun des fichiers choisis à l'aide de `FileReference.upload()`, comme le montre le code suivant :

```
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");
}
```

Comme l'événement `Event.COMPLETE` s'ajoute à chaque objet FileReference dans le tableau, Flash Player appelle la méthode `completeHandler()` à la fin du chargement de chacun des fichiers.

Exemple : création d'un client Telnet

L'exemple Telnet illustre les techniques de connexion à un serveur distant et de transmission des données à l'aide de la classe Socket. Cet exemple étudie les techniques suivantes :

- Création d'un client Telnet personnalisé à l'aide de la classe Socket
- Envoi de texte au serveur distant à l'aide de l'objet ByteArray
- Gestion des données reçues d'un serveur distant

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application Telnet se trouvent dans le dossier Samples/Telnet. L'application se compose des fichiers suivants :

Fichier	Description
TelnetSocket.fla ou TelnetSocket.mxml	Fichier d'application principal correspondant à l'interface utilisateur de Flex (MXML) ou Flash (FLA).
TelnetSocket.as	Classe Document fournissant la logique de l'interface utilisateur (Flash uniquement).
com/example/programmingas3/Telnet/Telnet.as	Fournit la fonctionnalité client Telnet à l'application, par exemple pour la connexion à un serveur distant et l'envoi, la réception et l'affichage des données.

Présentation de l'application socket Telnet

Le fichier principal TelnetSocket.mxml se charge de créer l'interface utilisateur (IU) de l'application entière.

Outre l'IU, ce fichier définit deux méthodes, `login()` et `sendCommand()`, qui permettent la connexion de l'utilisateur au serveur spécifié.

L'exemple suivant répertorie le code ActionScript du fichier d'application principal :

```
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 = "";
}
```

La première ligne de code importe la classe Telnet à partir du package personnalisé com.example.programmingas.socket. La deuxième déclare une occurrence de la classe Telnet, `telnetClient`, qui sera initialisée ultérieurement par la méthode `connect()`. Ensuite, la méthode `connect()` est déclarée et initialise la variable `telnetClient` déclarée auparavant. Cette méthode transmet le nom du serveur Telnet spécifié par l'utilisateur, le port de ce serveur et une référence à un composant TextArea dans la liste d'affichage, qui sert à afficher les réponses textuelles reçues du serveur socket. Les deux dernières lignes de la méthode `connect()` définissent la propriété `title` du composant Panel et active celui-ci, ce qui permet à l'utilisateur d'envoyer les données au serveur distant. La méthode finale du fichier d'application principal, `sendCommand()`, permet d'envoyer les commandes de l'utilisateur au serveur distant sous forme d'objet ByteArray.

Présentation de la classe Telnet

La classe Telnet se charge d'établir la connexion au serveur distant Telnet et d'envoyer et de recevoir les données.

La classe Telnet déclare les variables privées suivantes :

```
private var serverURL:String;
private var portNumber:int;
private var socket:Socket;
private var ta:TextArea;
private var state:int = 0;
```

La première variable, `serverURL`, contient l'adresse du serveur auquel se connecter, spécifiée par l'utilisateur.

La deuxième variable, `portNumber`, correspond au numéro de port sur lequel le serveur Telnet s'exécute actuellement. Par défaut, le service Telnet utilise le port 23.

La troisième variable, `socket`, est une occurrence de Socket qui essaiera d'établir une connexion au serveur défini par les variables `serverURL` et `portNumber`.

La quatrième variable, `ta`, est une référence à l'occurrence du composant TextArea sur la scène. Ce composant sert à afficher les réponses provenant du serveur Telnet distant ou les éventuels messages d'erreur.

La dernière variable, `state`, est une valeur numérique utilisée pour déterminer les options prises en charge par le client Telnet.

Comme vous l'avez vu précédemment, la fonction constructeur de la classe Telnet est appelée par la méthode `connect()` dans le fichier d'application principal.

Le constructeur Telnet prend trois paramètres : `server`, `port` et `output`. Les paramètres `server` et `port` spécifient le nom et le numéro de port du serveur Telnet. Le dernier paramètre, `output`, est une référence à l'occurrence du composant TextArea sur la scène, où s'afficheront les résultats du serveur pour l'utilisateur.

```
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();
    }
}
```

Ecriture de données dans un socket

Pour écrire des données sur une connexion Socket, vous pouvez appeler n'importe laquelle des méthodes d'écriture de la classe Socket (par exemple `writeBoolean()`, `writeByte()`, `writeBytes()` ou `writeDouble()`), puis purger les données dans le tampon de sortie à l'aide de la méthode `flush()`. Sur le serveur Telnet, les données sont écrites sur la connexion Socket à l'aide de la méthode `writeBytes()`, qui prend comme paramètre le tableau d'octets et l'envoie au tampon de sortie. La méthode `writeBytesToSocket()` se présente comme suit :

```
public function writeBytesToSocket(ba:ByteArray):void
{
    socket.writeBytes(ba);
    socket.flush();
}
```

Cette méthode est appelée par la méthode `sendCommand()` du fichier d'application principal.

Affichage des messages provenant du serveur socket

Dès qu'un message est reçu du serveur socket ou qu'un événement survient, la méthode personnalisée `msg()` est appelée. Cette méthode ajoute une chaîne au composant TextArea sur la scène et appelle une méthode `setScroll()` personnalisée, qui provoque le défilement vers le bas du composant TextArea. La méthode `msg()` se présente comme suit :

```
private function msg(value:String):void
{
    ta.text += value;
    setScroll();
}
```

Si vous n'appliquez pas le défilement automatique au contenu du composant TextArea, les utilisateurs auront à le faire manuellement pour consulter la dernière réponse du serveur.

Défilement du contenu d'un composant TextArea

La méthode `setScroll()` contient une seule ligne de code ActionScript qui permet de faire défiler verticalement le contenu du composant TextArea de manière que l'utilisateur puisse voir la dernière ligne de texte renvoyé. L'extrait de code suivant illustre la méthode `setScroll()` :

```
public function setScroll():void
{
    ta.verticalScrollPosition = ta.maxVerticalScrollPosition;
}
```

Cette méthode définit la propriété `verticalScrollPosition`, qui correspond au numéro de la ligne de caractère supérieure actuellement affichée, puis lui attribue la valeur de la propriété `maxVerticalScrollPosition`.

Exemple : chargement et téléchargement de fichiers

L'exemple FileIO présente les techniques de chargement et de téléchargement de fichiers dans Flash Player. Ces techniques sont les suivantes :

- Téléchargement de fichiers vers l'ordinateur de l'utilisateur
- Chargement de fichiers de l'ordinateur de l'utilisateur sur un serveur
- Annulation d'un téléchargement en cours
- Annulation d'un chargement en cours

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application FileIO se trouvent dans le dossier Samples/FileIO. L'application se compose des fichiers suivants :

Fichier	Description
FileIO.fla ou FileIO.mxml	Le fichier d'application principal dans Flash (FLA) ou Flex (MXML).
com/example/programmingas3/fileio/FileDownload.as	Classe incluant les méthodes de téléchargement de fichiers à partir d'un serveur.
com/example/programmingas3/fileio/FileUpload.as	Classe incluant les méthodes de chargement de fichiers sur un serveur.

Présentation de l'application FileIO

L'application FileIO contient l'interface utilisateur qui permet de charger ou télécharger des fichiers à l'aide de Flash Player. L'application commence par définir deux composants personnalisés, FileUpload et FileDownload, qui se trouvent dans le package com.example.programmingas3.fileio. Une fois que chaque composant a distribué son événement `contentComplete`, la méthode `init()` de chacun est appelée et transmet les références à des occurrences de composant ProgressBar et Button, qui permettent à l'utilisateur de suivre la progression du chargement/téléchargement ou d'annuler le transfert de fichier en cours.

Le code en question dans le fichier FileIO.mxml se présente comme suit (notez que dans la version Flash, le fichier FLA contient des composants placés sur la scène, dont les noms correspondent à ceux des composants Flex décrits dans cette étape) :

```
<example:FileUpload id="fileUpload" creationComplete="fileUpload.init(uploadProgress,
cancelUpload);" />
<example:FileDownload id="fileDownload"
creationComplete="fileDownload.init(downloadProgress, cancelDownload);" />
```

Le code suivant correspond au panneau de chargement du fichier, qui contient une barre de progression et deux boutons. Le premier bouton, `startUpload`, appelle la méthode `FileUpload.startUpload()`, qui elle-même appelle la méthode `FileReference.browse()`. L'extrait suivant présente le code du panneau de chargement du fichier :

```
<mx:Panel title="Upload File" paddingTop="10" paddingBottom="10" paddingLeft="10"
paddingRight="10">
    <mx:ProgressBar id="uploadProgress" label="" mode="manual" />
    <mx:ControlBar horizontalAlign="right">
        <mx:Button id="startUpload" label="Upload..." click="fileUpload.startUpload();" />
        <mx:Button id="cancelUpload" label="Cancel" click="fileUpload.cancelUpload();"
enabled="false" />
    </mx:ControlBar>
</mx:Panel>
```

Ce code place une occurrence de composant ProgressBar et deux occurrences de composant Button sur la scène. Lorsque l'utilisateur clique sur le bouton de chargement `startUpload`, une boîte de dialogue du système d'exploitation s'affiche pour permettre à l'utilisateur de sélectionner le fichier à charger sur le serveur distant. L'autre bouton, `cancelUpload`, est désactivé par défaut ; cependant, lorsque l'utilisateur commence le chargement du fichier, ce bouton devient actif et lui permet de mettre fin à tout moment au transfert du fichier.

Le code du panneau de téléchargement du fichier est le suivant :

```
<mx:Panel title="Download File" paddingTop="10" paddingBottom="10" paddingLeft="10"
paddingRight="10">
    <mx:ProgressBar id="downloadProgress" label="" mode="manual" />
    <mx:ControlBar horizontalAlign="right">
        <mx:Button id="startDownload" label="Download..."
click="fileDownload.startDownload();" />
        <mx:Button id="cancelDownload" label="Cancel" click="fileDownload.cancelDownload();"
enabled="false" />
    </mx:ControlBar>
</mx:Panel>
```

Ce code est très semblable au code de chargement de fichier. Lorsque l'utilisateur clique sur le bouton de téléchargement `startDownload`, la méthode `FileDownload.startDownload()` est appelée. Celle-ci débute le téléchargement du fichier spécifié dans la variable `FileDownload.DOWNLOAD_URL`. Pendant le téléchargement du fichier, la barre de progression s'actualise pour indiquer le pourcentage du fichier déjà téléchargé. L'utilisateur peut annuler l'opération à tout moment en cliquant sur le bouton `cancelDownload`, qui arrête immédiatement le téléchargement de fichier en cours.

Téléchargement de fichiers à partir d'un serveur distant

Le téléchargement de fichier à partir d'un serveur distant est géré par la classe flash.net.FileReference et la classe personnalisée com.example.programmingas3.fileio.FileDownload. Lorsque l'utilisateur clique sur le bouton de téléchargement, Flash Player commence à télécharger le fichier spécifié dans la variable `DOWNLOAD_URL` de la classe FileDownload.

La classe FileDownload commence par définir quatre variables au sein du package com.example.programmingas3.fileio, comme le montre le code suivant :

```
/**
 * Hard-code the URL of file to download to user's computer.
 */
private const DOWNLOAD_URL:String = "http://www.yourdomain.com/file_to_download.zip";

/**
 * Create a FileReference instance to handle the file download.
 */
private var fr:FileReference;

/**
 * Define reference to the download ProgressBar component.
 */
private var pb:ProgressBar;

/**
 * Define reference to the "Cancel" button which will immediately stop
 * the current download in progress.
 */
private var btn:Button;
```

La première variable, `DOWNLOAD_URL`, contient le chemin d'accès au fichier qui est téléchargé sur l'ordinateur de l'utilisateur lorsque celui-ci clique sur le bouton de téléchargement dans le fichier d'application principal.

La deuxième variable, `fr`, est un objet FileReference qui est initialisé dans la méthode `FileDownload.init()` et gérera le téléchargement du fichier distant sur l'ordinateur de l'utilisateur.

Les deux dernières variables, `pb` et `btn`, contiennent les références aux occurrences de composant ProgressBar et Button sur la scène, qui sont initialisées par la méthode `FileDownload.init()`.

Initialisation du composant FileDownload

Le composant FileDownload est initialisé par l'appel de la méthode `init()` de la classe FileDownload. Cette méthode prend deux paramètres, `pb` et `btn`, qui sont des occurrences de composant ProgressBar et Button, respectivement.

Le code de la méthode `init()` se présente comme suit :

```
/**
 * Set references to the components, and add listeners for the OPEN,
 * PROGRESS, and COMPLETE events.
 */
public function init(pb:ProgressBar, btn:Button):void
{
    // Set up the references to the progress bar and cancel button,
    // which are passed from the calling script.
    this.pb = pb;
    this.btn = btn;

    fr = new FileReference();
    fr.addEventListener(Event.OPEN, openHandler);
    fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
    fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

Lancement du téléchargement d'un fichier

Lorsque l'utilisateur clique sur le bouton de téléchargement sur la scène, la méthode `startDownload()` doit initialiser le processus de téléchargement de fichier. L'extrait suivant illustre la méthode `startDownload()` :

```
/**
 * Begin downloading the file specified in the DOWNLOAD_URL constant.
 */
public function startDownload():void
{
    var request:URLRequest = new URLRequest();
    request.url = DOWNLOAD_URL;
    fr.download(request);
}
```

Tout d'abord, la méthode `startDownload()` crée un nouvel objet URLRequest et définit l'URL cible de la valeur spécifiée par la variable `DOWNLOAD_URL`. Ensuite, la méthode `FileReference.download()` est appelée et l'objet URLRequest créé est transmis comme paramètre. Le système d'exploitation affiche alors une boîte de dialogue qui invite l'utilisateur à sélectionner un emplacement cible pour le document demandé. Une fois l'emplacement choisi, l'événement `open` (`Event.OPEN`) est transmis et la méthode `openHandler()` appelée.

La méthode `openHandler()` définit le format de texte de la propriété `label` du composant ProgressBar et active le bouton d'annulation, qui permet à l'utilisateur d'arrêter immédiatement le téléchargement en cours. La méthode `openHandler()` se présente comme suit :

```
/**
 * When the OPEN event has dispatched, change the progress bar's label
 * and enable the "Cancel" button, which allows the user to abort the
 * download operation.
 */
private function openHandler(event:Event):void
{
    pb.label = "DOWNLOADING %3%%";
    btn.enabled = true;
}
```

Surveillance de la progression du téléchargement d'un fichier

Pendant le téléchargement d'un fichier du serveur distant sur l'ordinateur de l'utilisateur, l'événement `progress` (`ProgressEvent.PROGRESS`) est distribué à intervalles réguliers. Dès distribution de l'événement `progress`, la méthode `progressHandler()` est appelée et le composant ProgressBar sur la scène est actualisé. Le code de la méthode `progressHandler()` se présente comme suit :

```
/**
 * While the file is downloading, update the progress bar's status.
 */
private function progressHandler(event:ProgressEvent):void
{
    pb.setProgress(event.bytesLoaded, event.bytesTotal);
}
```

L'événement Progress contient deux propriétés, `bytesLoaded` et `bytesTotal`, qui servent à actualiser le composant ProgressBar sur la scène. L'utilisateur reçoit ainsi une indication de la quantité de données déjà téléchargée et du restant. Il peut mettre fin au transfert de fichier à tout moment en cliquant sur le bouton d'annulation situé sous la barre de progression.

Si le téléchargement réussit, l'événement `complete` (`Event.COMPLETE`) appelle la méthode `completeHandler()`, qui avertit l'utilisateur de la fin de l'opération et désactive le bouton d'annulation. Le code de la méthode `completeHandler()` se présente comme suit :

```
/**
 * Once the download has completed, change the progress bar's label one
 * last time and disable the "Cancel" button since the download is
 * already completed.
 */
private function completeHandler(event:Event):void
{
    pb.label = "DOWNLOAD COMPLETE";
    btn.enabled = false;
}
```

Annulation du téléchargement d'un fichier

L'utilisateur peut à tout moment mettre fin au transfert de fichiers et empêcher le téléchargement d'octets supplémentaires en cliquant sur le bouton d'annulation de la scène. L'extrait suivant présente le code d'annulation du téléchargement :

```
/**
 * Cancel the current file download.
 */
public function cancelDownload():void
{
    fr.cancel();
    pb.label = "DOWNLOAD CANCELLED";
    btn.enabled = false;
}
```

Tout d'abord, le code arrête immédiatement le transfert de fichiers, empêchant ainsi le téléchargement de davantage de données. Ensuite, la propriété Label de la barre de progression s'actualise pour confirmer à l'utilisateur l'annulation du téléchargement. Enfin, le bouton d'annulation est désactivé, pour empêcher l'utilisateur de cliquer de nouveau sur son entrée tant qu'une autre tentative de téléchargement n'est pas lancée.

Chargement de fichiers sur un serveur distant

Le processus de chargement de fichier est très semblable au téléchargement. La classe FileUpload déclare les quatre mêmes variables, comme le montre le code suivant :

```
private const UPLOAD_URL:String = "http://www.yourdomain.com/your_upload_script.cfm";
private var fr:FileReference;
private var pb:ProgressBar;
private var btn:Button;
```

A la différence de la variable `FileDownload.DOWNLOAD_URL`, `UPLOAD_URL` contient l'URL du script côté serveur qui chargera le fichier à partir de l'ordinateur de l'utilisateur. Les trois autres variables ont le même comportement que leurs équivalents dans la classe FileDownload.

Initialisation du composant FileUpload

Le composant FileUpload contient une méthode `init()`, qui est appelée à partir de l'application principale. Cette méthode prend deux paramètres, `pb` et `btn`, qui sont des références aux composants ProgressBar et Button sur la scène. Ensuite, la méthode `init()` initialise l'objet FileReference défini plus tôt par la classe FileUpload. Enfin, la méthode attribue quatre écouteurs d'événement à l'objet FileReference. Le code de la méthode `init()` se présente comme suit :

```
public function init(pb:ProgressBar, btn:Button):void
{
    this.pb = pb;
    this.btn = btn;

    fr = new FileReference();
    fr.addEventListener(Event.SELECT, selectHandler);
    fr.addEventListener(Event.OPEN, openHandler);
    fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
    fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

Lancement du chargement d'un fichier

Le chargement du fichier est initialisé lorsque l'utilisateur clique sur le bouton de chargement de la scène, qui appelle la méthode `FileUpload.startUpload()`. Celle-ci appelle la méthode `browse()` de la classe FileReference qui entraîne l'affichage par le système d'exploitation d'une boîte de dialogue invitant l'utilisateur à sélectionner le fichier à charger sur le serveur distant. L'extrait suivant présente le code de la méthode `startUpload()` :

```
public function startUpload():void
{
    fr.browse();
}
```

Une fois que l'utilisateur à sélectionner le fichier, l'événement `select` (`Event.SELECT`) est distribué, ce qui entraîne l'appel de la méthode `selectHandler()`. La méthode `selectHandler()` créé un objet URLRequest et attribue à la propriété `URLRequest.url` la valeur de la constante `UPLOAD_URL` définie plus haut dans le code. Enfin, l'objet FileReference charge le fichier sélectionné dans le script côté serveur spécifié. Le code de la méthode `selectHandler()` se présente comme suit :

```
private function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest();
    request.url = UPLOAD_URL;
    fr.upload(request);
}
```

Le reste du code de la classe FileUpload est identique au code défini dans la classe FileDownload. Si l'utilisateur souhaite mettre fin au chargement, il peut cliquer à tout moment sur le bouton d'annulation qui définit le libellé de la barre de progression et arrête immédiatement le transfert du fichier. La barre de progression est actualisée dès que l'événement `progress` (`ProgressEvent.PROGRESS`) est distribué. De même, une fois que le chargement est terminé, la barre de progression s'actualise pour confirmer à l'utilisateur la réussite du chargement. Le bouton d'annulation est alors désactivé jusqu'à ce que l'utilisateur lance un nouveau transfert de fichier.

Chapitre 28 : Environnement du système client

Ce chapitre explique comment interagir avec le système utilisateur. Il vous montre comment déterminer les fonctions prises en charge et comment construire des fichiers SWF multilingues à l'aide de l'éditeur de la méthode d'entrée installée (IME) de l'utilisateur (le cas échéant). Il vous présente enfin les utilisations typiques des domaines d'application.

Principes de base de l'environnement du système client

Introduction à l'environnement du système client

Au fur et à mesure que vous créez des applications ActionScript plus avancées, il se peut que vous souhaitiez en savoir plus sur les systèmes d'exploitation de vos utilisateurs et leurs fonctions d'accès. L'environnement du système client est un ensemble de classes présentes dans le package flash.system qui permettent d'accéder à des fonctionnalités au niveau du système, telles que :

- Détermination de l'application et du domaine de sécurité dans lesquels un SWF est exécuté
- Détermination des fonctionnalités de l'occurrence de Flash® Player ou Adobe® AIR™ de l'utilisateur (taille de l'écran - résolution) et des fonctionnalités disponibles (audio MP3, par exemple)
- Création de sites multilingues utilisant l'IME
- Interaction avec le conteneur de Flash Player (qui doit être une page HTML ou une application de conteneur) ou d'AIR
- Enregistrement d'informations dans le presse-papiers de l'utilisateur

Le package flash.system comprend aussi les classes IMEConversionMode et SecurityPanel. Ces classes contiennent des constantes statiques que vous pouvez utiliser avec les classes IME et de sécurité, respectivement.

Tâches courantes de l'environnement du système client

Les tâches courantes suivantes qui utilisent le système client avec ActionScript sont décrites dans ce chapitre :

- Détermination de la quantité de mémoire utilisée par votre application
- Copie de texte dans le presse-papiers de l'utilisateur
- Détermination des fonctionnalités de l'ordinateur de l'utilisateur telles que :
  - Résolution de l'écran, couleur, PPP et proportions en pixel
  - Système d'exploitation
  - Prise en charge pour les sons en flux continu, la vidéo en flux continu et la lecture mp3
  - Vérification de la version de Flash Player installée pour savoir s'il s'agit d'une version de débogage
- Utilisation de domaines d'application:
  - Définition d'un domaine d'application
  - Séparation de code de fichiers SWF en domaines d'application

- Utilisation d'un IME dans votre application :
  - Déterminer si un IME est installé
  - Déterminer le mode de conversion IME et le définir
  - Désactiver l'IME pour des champs de texte
  - Détecter lorsqu'une conversion IME a lieu

Concepts importants et terminologie

La liste de référence suivante contient les termes importants utilisés dans ce chapitre :

- Système d'exploitation : programme principal qui est exécuté sur un ordinateur, dans lequel toutes les autres applications sont exécutées (Microsoft Windows, Mac OS X ou Linux®, par exemple).
- Presse-papiers : conteneur du système d'exploitation qui contient du texte ou des éléments qui sont copiés ou coupés, et à partir duquel des éléments sont collés dans des applications.
- Domaine d'application : mécanisme permettant de séparer les classes utilisées dans différents fichiers SWF de façon à ce que si les fichiers SWF comprennent différentes classes ayant le même nom, elles ne soient pas supprimées.
- IME : programme (ou outil de système d'exploitation) utilisé pour entrer des caractères complexes ou des symboles utilisant un clavier standard.
- Système client : en termes de programmation, un *client* est la partie d'une application (ou l'application entière) exécutée sur un ordinateur et utilisée par un seul utilisateur. Le *système client* est le système d'exploitation sous-jacent sur l'ordinateur de l'utilisateur.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester des exemples de code. Tous les codes fournis dans ce chapitre comprennent l'appel de la fonction `trace()` approprié pour écrire les valeurs testées. Pour tester les codes de ce chapitre :

1. Créez un document Flash vide.
2. Sélectionnez une image-clé dans le scénario.
3. Ouvrez le panneau Actions et copiez le code dans le panneau Script.
4. Exécutez le programme en sélectionnant Contrôle >Tester l'animation.

   Les résultats des fonctions `trace()` des codes s'affichent dans le panneau Sortie.

Certains des prochains codes sont plus complexes et sont écrits sous la forme d'une classe. Pour tester ces exemples :

1. Créez un document Flash vide et enregistrez-le sur votre ordinateur.
2. Créez un nouveau fichier ActionScript et enregistrez-le dans le même répertoire que le document Flash. Le nom du fichier doit correspondre au nom de la classe du code. Par exemple, si le code définit une classe SystemTest, enregistrez le fichier ActionScript sous le nom SystemTest.as.
3. Copiez le code dans le fichier ActionScript et enregistrez le fichier.
4. Dans le document Flash, cliquez sur une partie vide de la scène ou de l'espace de travail pour activer l'Inspecteur des propriétés du document.
5. Dans l'Inspecteur des propriétés, dans le champ Classe du document, saisissez le nom de la classe ActionScript que vous avez copiée du texte.
6. Exécutez le programme en sélectionnant Contrôle >Tester l'animation.

Les résultats de l'exemple s'affichent dans le panneau Sortie.

Vous trouverez plus de détails sur les techniques de test des codes à la rubrique. « Test des exemples de code contenus dans un chapitre » à la page 36.

Utilisation de la classe System

La classe System contient des méthodes et des propriétés qui vous permettent d'interagir avec le système d'exploitation de l'utilisateur et de récupérer l'utilisation mémoire actuelle pour Flash Player ou AIR. Les méthodes et les propriétés de la classe System vous permettent aussi d'écouter les événements `imeComposition`, d'indiquer à Flash Player ou AIR de charger des fichiers texte externes à l'aide de la page de code active de l'utilisateur ou d'Unicode, ou de définir le contenu du presse-papiers de l'utilisateur.

Obtention de données sur le système de l'utilisateur pendant l'exécution

En vérifiant la propriété `System.totalMemory`, vous pouvez déterminer la quantité de mémoire (en octets) que Flash Player ou AIR utilise actuellement. Cette propriété vous permet de surveiller l'utilisation mémoire et d'optimiser vos applications en fonction de ses variations. Par exemple, si un effet visuel particulier utilise une importante quantité de mémoire, vous pouvez envisager de le modifier ou de le supprimer entièrement.

La propriété `System.ime` est une référence à l'IME actuellement installé. Elle vous permet d'écouter les événements `imeComposition` (`flash.events.IMEEvent.IME_COMPOSITION`) à l'aide de la méthode `addEventListener()`.

La troisième propriété dans la classe System est `useCodePage`. Lorsque `useCodePage` est défini sur `true`, Flash Player et AIR utilisent la page de code traditionnelle du système d'exploitation qui exécute le lecteur pour charger des fichiers texte. Si vous lui attribuez la valeur `false`, vous indiquez à Flash Player ou AIR d'interpréter le fichier externe avec Unicode.

Si vous définissez `System.useCodePage` sur `true`, souvenez-vous que la page de code classique du système d'exploitation exécutant le lecteur doit inclure les caractères utilisés dans votre fichier texte externe afin d'afficher le texte. Par exemple, si vous chargez un fichier texte externe contenant des caractères chinois, ceux-ci ne peuvent s'afficher sur un système qui utilise la page de code anglaise de Windows car elle ne comprend pas les caractères chinois.

Pour que les utilisateurs de toutes les plates-formes puissent afficher les fichiers texte externes utilisés dans vos fichiers SWF, vous devez coder tous les fichiers texte externes en Unicode et conserver la valeur par défaut `false` de la propriété `System.useCodePage`. Ainsi, Flash Player et AIR interprètent le texte comme de l'Unicode.

Enregistrement du texte dans le presse-papiers

La classe System inclut une méthode appelée `setClipboard()` qui permet à Flash Player et AIR de placer une chaîne spécifique dans le presse-papiers de l'utilisateur. Pour des raisons de sécurité, il n'existe pas de méthode `Security.getClipboard()` car elle donnerait la possibilité d'accéder aux dernières données copiées dans le presse-papiers de l'utilisateur.

Le code suivant illustre comment un message d'erreur peut être copié dans le presse-papiers de l'utilisateur lorsqu'une erreur de sécurité survient. Le message d'erreur permettra à l'utilisateur de signaler un bogue potentiel dans une application.

```
private function securityErrorHandler(event:SecurityErrorEvent):void
{
    var errorString:String = "[" + event.type + "] " + event.text;
    trace(errorString);
    System.setClipboard(errorString);
}
```

Utilisation de la classe Capabilities

La classe Capabilities permet aux développeurs de déterminer l'environnement dans lequel le fichier SWF est exécuté. A l'aide des diverses propriétés de la classe Capabilities, vous pouvez déterminer la résolution et la langue du système de l'utilisateur, savoir si ce système prend en charge les logiciels d'accessibilité et identifier la version de Flash Player ou AIR actuellement installée.

La vérification des propriétés de la classe Capabilities vous autorise à personnaliser votre application pour un fonctionnement optimal sur l'environnement de l'utilisateur. Par exemple, si vous vérifiez les propriétés `Capabilities.screenResolutionX` et `Capabilities.screenResolutionY`, vous pouvez déterminer la résolution d'affichage du système de l'utilisateur et décider de la taille de vidéo la plus appropriée. Vous pouvez aussi vérifier la propriété `Capabilities.hasMP3` pour voir si le système de l'utilisateur prend en charge la lecture du format mp3 avant d'essayer de charger un fichier mp3 externe.

Le code ci-après utilise une expression régulière pour analyser la version de Flash Player utilisée sur le client :

```
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.");
}
```

Si vous souhaitez envoyer les capacités du système de l'utilisateur à un script côté serveur afin de stocker les informations dans une base de données, vous pouvez utiliser le code ActionScript suivant :

```
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);
```

Utilisation de la classe ApplicationDomain

Le rôle de la classe ApplicationDomain est de stocker un tableau des définitions ActionScript 3.0. L'ensemble du code d'un fichier SWF est défini de sorte à exister dans un domaine d'application. Les domaines d'application vous servent à partitionner les classes qui se trouvent dans un même domaine de sécurité. Ainsi, plusieurs définitions de la même classe peuvent exister et les enfants peuvent réutiliser les définitions des parents.

Vous pouvez utiliser les domaines d'application lors du chargement, au moyen de la classe Loader, d'un fichier SWF externe écrit en ActionScript 3.0. (Notez que vous ne pouvez pas utiliser les domaines d'application lorsque vous chargez une image ou un fichier SWF écrit en ActionScript 1.0 ou 2.0.) Toutes les définitions ActionScript 3.0 contenues dans la classe chargée sont stockées dans le domaine d'application. Lorsque vous chargez un fichier SWF, vous devez indiquer que le fichier doit être inclus dans le même domaine d'application que l'objet Loader en attribuant au paramètre `applicationDomain` de l'objet LoaderContext la valeur `ApplicationDomain.currentDomain`. Si vous placez le fichier SWF chargé dans le même domaine d'application, vous pourrez accéder directement à ses classes. Cela s'avère pratique si vous chargez un fichier SWF qui contient des médias incorporés auxquels vous pouvez accéder via les noms de classe associés, ou si vous voulez accéder aux méthodes du fichier SWF chargé, comme le montre l'exemple suivant :

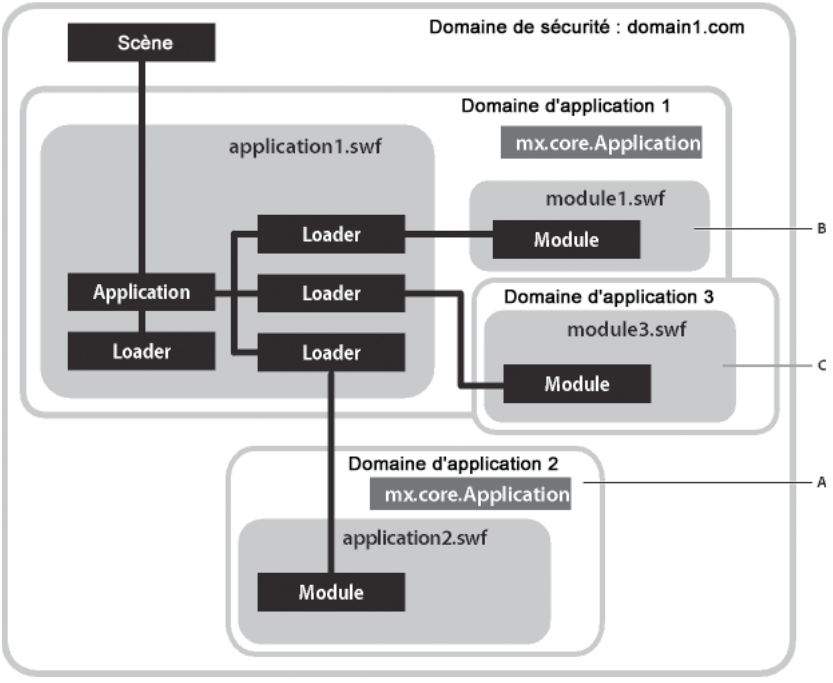
```
package
{
    import flash.display.Loader;
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLRequest;
    import flash.system.ApplicationDomain;
    import flash.system.LoaderContext;

    public class ApplicationDomainExample extends Sprite
    {
        private var ldr:Loader;
        public function ApplicationDomainExample()
        {
            ldr = new Loader();
            var req:URLRequest = new URLRequest("Greeter.swf");
            var ldrContext:LoaderContext = new LoaderContext(false,
ApplicationDomain.currentDomain);
            ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, completeHandler);
            ldr.load(req, ldrContext);
        }
        private function completeHandler(event:Event):void
        {
            ApplicationDomain.currentDomain.getDefinition("Greeter");
            var myGreeter:Greeter = Greeter(event.target.content);
            var message:String = myGreeter.welcome("Tommy");
            trace(message); // Hello, Tommy
        }
    }
}
```

Voici d'autres points à garder à l'esprit lorsque vous utilisez les domaines d'application :

- L'ensemble du code d'un fichier SWF est défini de sorte à exister dans un domaine d'application. Votre application principale s'exécute dans le *domaine d'application* en cours. Le *domaine du système* contient tous les domaines d'application, y compris le domaine en cours ; il contient donc toutes les classes Flash Player.

- A l'exception du domaine du système, tous les domaines d'application sont associés à un domaine parent. Le domaine parent du domaine de votre application principale est le domaine du système. Les classes chargées ne sont définies que si leur parent ne les définit pas encore. Vous ne pouvez pas remplacer une définition de classe chargée par une définition plus récente.

Le schéma suivant figure une application qui charge du contenu à partir de divers fichiers SWF au sein d'un domaine unique, domain1.com. Selon le contenu chargé, différents domaines d'application peuvent être utilisés. Le texte suivant décrit la logique utilisée pour définir le domaine d'application approprié pour chaque fichier SWF de l'application.



A. *Utilisation A* ***B.*** *Utilisation B* ***C.*** *Utilisation C*

Le fichier principal d'application est application1.swf. Il contient des objets Loader qui chargent du contenu à partir d'autres fichiers SWF. Dans ce scénario, le domaine en cours est Application domain 1. Utilisation A, Utilisation B et Utilisation C illustrent les différentes techniques permettant de définir le domaine d'application approprié pour chaque fichier SWF de l'application.

Utilisation A Partitionnez le fichier SWF enfant en créant un enfant du domaine du système. Dans le schéma, le domaine d'application 2 est créé comme enfant du domaine du système. Le fichier application2.swf est chargé dans le domaine d'application 2 et ses définitions de classe sont ainsi partitionnées à partir des classes définies dans application1.swf.

Cette technique s'appliquera par exemple lorsqu'une ancienne application doit charger dynamiquement une nouvelle version de la même application sans créer de conflits. Les conflits sont éliminés parce que même si les noms de classe sont les mêmes, ils sont répartis dans différents domaines d'application.

Le code suivant crée un domaine d'application qui est un enfant du domaine du système, puis commence à charger un fichier SWF à l'aide de ce domaine d'application :

```
var appDomainA:ApplicationDomain = new ApplicationDomain();

var contextA:LoaderContext = new LoaderContext(false, appDomainA);
var loaderA:Loader = new Loader();
loaderA.load(new URLRequest("application2.swf"), contextA);
```

Utilisation B Ajoutez de nouvelles définitions de classe aux définitions actuelles. Le domaine d'application module1.swf est défini sur le domaine actif (application domain 1). Vous pouvez alors ajouter au jeu actuel de définitions de classe de l'application de nouvelles définitions de classe. Cela pourrait servir pour une bibliothèque d'exécution partagée appartenant à l'application principale. Le fichier SWF est traité comme une bibliothèque partagée distante (RSL, remote shared library). Utilisez cette technique pour charger des RSL à l'aide d'un fichier de préchargement avant le lancement de l'application.

Le code suivant charge un fichier SWF, en définissant son domaine d'application sur le domaine actif :

```
var appDomainB:ApplicationDomain = ApplicationDomain.currentDomain;

var contextB:LoaderContext = new LoaderContext(false, appDomainB);
var loaderB:Loader = new Loader();
loaderB.load(new URLRequest("module1.swf"), contextB);
```

Utilisation C Utilisez les définitions de classe du parent en ajoutant un nouveau domaine enfant au domaine actif. Le domaine d'application de module3.swf est un enfant du domaine actuel, qui utilise pour toutes les classes les versions du parent. Cette technique peut s'appliquer à un module d'une application Internet riche (RIA, Rich Internet Application) à plusieurs écrans, qui serait chargé comme enfant de l'application principale et utiliserait les types de cette dernière. Si vous pouvez garantir que toutes les classes sont toujours mises à jour pour rester compatibles avec les anciennes versions et que l'application de chargement est toujours plus récente que les contenus qu'elle charge, les enfants utiliseront les versions des parents. L'utilisation d'un nouveau domaine d'application vous permet de décharger toutes les définitions de classe en vue du nettoyage, à condition de veiller à ce qu'il ne subsiste aucune référence au fichier SWF enfant.

Cette technique autorise les modules chargés à partager les objets Singleton et les membres de classe statiques de l'objet Loader.

Le code suivant crée un domaine enfant dans le domaine actif et commence à charger un fichier SWF à l'aide de ce domaine d'application :

```
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);
```

Utilisation de la classe IME

La classe IME permet de manipuler l'IME du système d'exploitation à partir de Flash Player ou Adobe AIR.

A l'aide d'ActionScript, vous pouvez déterminer les éléments suivants :

- Si un IME est installé sur l'ordinateur de l'utilisateur (`Capabilities.hasIME`)
- Si l'IME est activé ou désactivé sur l'ordinateur de l'utilisateur (`IME.enabled`)
- Le mode de conversion utilisé par l'IME actif (`IME.conversionMode`)

Vous pouvez associer un champ de saisie de texte à un contexte IME particulier. Lorsque vous passez d'un champ de saisie à un autre, vous pouvez également changer l'IME pour utiliser les caractères Hiragana (japonais), des nombres à pleine chasse, des nombres à demi-chasse, la saisie directe, etc.

Un IME permet aux utilisateurs d'entrer des caractères de texte non ASCII des langues codées sur plusieurs octets, telles que le chinois, le japonais et le coréen.

Pour plus d'informations sur les IME, reportez-vous à la documentation du système d'exploitation correspondant à la plate-forme pour laquelle vous développez l'application. Pour davantage de ressources, consultez également les sites Web suivants :

- http://www.msdn.microsoft.com/goglobal/
- http://developer.apple.com/documentation/
- http://www.java.sun.com/

Remarque : *si aucun IME n'est actif sur l'ordinateur de l'utilisateur, tout appel aux méthodes ou propriétés IME, autres que* `Capabilities.hasIME`*, échoue. Lorsque vous activez manuellement un IME, les appels ActionScript suivants aux méthodes et aux propriétés IME fonctionnent comme prévu. Par exemple, si vous utilisez un IME japonais, vous devez l'activer avant d'appeler une méthode ou une propriété IME.*

Confirmation de l'installation et de l'activation d'un IME

Avant d'appeler des méthodes ou propriétés IME, vous devez toujours vérifier si un IME est installé et activé sur l'ordinateur de l'utilisateur. Le code suivant montre comment vérifier que l'utilisateur dispose d'un IME installé et activé avant d'appeler une méthode :

```
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.");
}
```

Le code précédent commence par vérifier si un IME est installé à l'aide la propriété `Capabilities.hasIME`. Si la valeur de la propriété est `true`, le code vérifie ensuite si l'IME est activé à l'aide de la propriété `IME.enabled`.

Identification du mode de conversion IME activé

Lorsque vous construisez une application multilingue, il peut être nécessaire de déterminer le mode de conversion actif dans le système d'exploitation. Le code suivant montre comment vérifier si l'utilisateur dispose d'un IME installé et, le cas échéant, quel mode de conversion IME est activé :

```
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.";
}
```

Le code ci-dessus commence par vérifier si l'utilisateur dispose d'un IME. Ensuite, il vérifie le mode de conversion actuellement utilisé par l'IME en comparant la propriété `IME.enabled` à chacune des constantes de la classe IMEConversionMode.

Définition du mode de conversion IME

Lorsque vous modifiez le mode de conversion de l'IME de l'utilisateur, veillez à ce que le code soit enveloppé dans un bloc `try..catch`, car la définition du mode de conversion à l'aide de la propriété `conversionMode` peut donner lieu à une erreur si l'IME ne peut pas définir le mode choisi. Le code suivant illustre l'utilisation d'un bloc `try..catch` lors de la définition de la propriété`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;
    }
}
```

Ce code commence par créer un champ de texte qui sert à afficher un message d'état à l'intention de l'utilisateur. Ensuite, si l'IME est installé, le code l'active et définit le mode de conversion coréen. Si l'ordinateur de l'utilisateur ne dispose pas d'un IME coréen, une erreur est renvoyée par Flash Player ou AIR et interceptée par le bloc `try..catch`. Le bloc `try..catch` affiche le message d'erreur dans le champ de texte créé précédemment.

Désactivation de l'IME pour certains champs de texte

Dans certains cas, il peut être nécessaire de désactiver l'IME de l'utilisateur pendant que ce dernier saisit des caractères. Par exemple, si un champ de texte accepte uniquement des caractères numériques, il peut être préférable d'éviter l'intervention de l'IME pour ne pas ralentir la saisie des données.

L'exemple suivant montre comment écouter les événements `FocusEvent.FOCUS_IN` et `FocusEvent.FOCUS_OUT` et désactiver l'IME de l'utilisateur en conséquence :

```
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;
    }
}
```

Cet exemple crée deux champs de saisie de texte, `phoneTxt` et `nameTxt`, puis ajoute deux écouteurs d'événement au champ `phoneTxt`. Lorsque l'utilisateur place le focus sur le champ `phoneTxt`, un événement `FocusEvent.FOCUS_IN` est distribué et l'IME est désactivé. Lorsque le focus est retiré du champ `phoneTxt`, l'événement `FocusEvent.FOCUS_OUT` est distribué pour réactiver l'IME.

Ecoute des événements IME composition

Les événements IME composition sont distribués lors de la définition d'une chaîne de composition. Par exemple, si l'IME de l'utilisateur est activé et que l'utilisateur saisit une chaîne en japonais, l'événement `IMEEvent.IME_COMPOSITION` est distribué dès que l'utilisateur sélectionne la chaîne de composition. Pour écouter l'événement `IMEEvent.IME_COMPOSITION`, vous devez ajouter un écouteur d'événement à la propriété statique `ime` de la classe System (`flash.system.System.ime.addEventListener(...)`), comme le montre l'exemple suivant :

```
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;
}
```

Le code ci-dessus crée deux champs de texte et les ajoute à la liste d'affichage. Le premier champ, `inputTxt`, est un champ de saisie de texte qui permet à l'utilisateur d'entrer du texte japonais. Le second champ, `outputTxt`, est un champ de texte dynamique qui affiche des messages d'erreur à l'utilisateur ou reprend la chaîne japonaise que l'utilisateur saisit dans le champ `inputTxt`.

Exemple : détection des capacités du système

L'exemple CapabilitiesExplorer vous montre comment utiliser la classe flash.system.Capabilities pour déterminer les fonctions prises en charge par la version de Flash Player ou AIR de l'utilisateur. Cet exemple étudie les techniques suivantes :

- Détection des fonctions prises en charge par la version de Flash Player ou AIR de l'utilisateur à l'aide de la classe Capabilities

- Utilisation de la classe ExternalInterface pour détecter les paramètres de navigation pris en charge par le navigateur de l'utilisateur

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers d'application CapabilitiesExplorer se trouvent dans le dossier Samples/CapabilitiesExplorer. Cette application se compose des fichiers suivants :

Fichier	Description
CapabilitiesExplorer.fla ou CapabilitiesExplorer.mxml	Le fichier d'application principal dans Flash (FLA) ou Flex (MXML).
com/example/programmingas3/capabilities/CapabilitiesGrabber.as	Classe fournissant la principale fonctionnalité de l'application, notamment l'ajout des capacités du système dans un tableau, le tri des éléments et l'extraction des capacités du navigateur à l'aide de la classe ExternalInterface.
capabilities.html	Conteneur HTML comprenant le code JavaScript nécessaire à la communication avec l'API externe.

Présentation de CapabilitiesExplorer

Le fichier CapabilitiesExplorer.mxml se charge de définir l'interface utilisateur de l'application CapabilitiesExplorer. Les capacités de la version de Flash Player ou AIR de l'utilisateur seront affichées dans une occurrence du composant DataGrid sur la scène. Les capacités du navigateur sont également affichées si l'application est exécutée à partir d'un conteneur HTML et si l'API externe est disponible.

Une fois que l'événement `creationComplete` du fichier de l'application principale est distribué, la méthode `initApp()` est appelée. La méthode `initApp()` appelle la méthode `getCapabilities()` à partir de la classe com.example.programmingas3.capabilities.CapabilitiesGrabber. Le code de la méthode `initApp()` se présente comme suit :

```
private function initApp():void
{
    var dp:Array = CapabilitiesGrabber.getCapabilities();
    capabilitiesGrid.dataProvider = dp;
}
```

La méthode `CapabilitiesGrabber.getCapabilities()` renvoie un tableau trié contenant les capacités de Flash Player ou AIR et du navigateur, qui est alors affecté à la propriété `dataProvider` de l'occurrence du composant DataGrid `capabilitiesGrid` sur la scène.

Présentation de la classe CapabilitiesGrabber

La méthode statique `getCapabilities()` de la classe CapabilitiesGrabber ajoute chaque propriété de la classe flash.system.Capabilities dans un tableau (`capDP`). Elle appelle ensuite la méthode statique `getBrowserObjects()` de la classe CapabilitiesGrabber. La méthode `getBrowserObjects()` utilise l'API externe pour passer en boucle sur l'objet navigator du navigateur, qui contient les capacités du navigateur. La méthode `getCapabilities()` se présente comme suit :

```
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;
}
```

La méthode `getBrowserObjects()` renvoie un tableau de toutes les propriétés de l'objet navigator du navigateur. Si ce tableau contient un élément ou plus, le tableau des capacités du navigateur (`navArr`) s'ajoute au tableau des capacités de Flash Player (`capDP`) et le tableau résultant est trié par ordre alphabétique. Enfin, le tableau trié est renvoyé au fichier de l'application principale, qui ensuite remplit la grille de données. Le code de la méthode `getBrowserObjects()` se présente comme suit :

```
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;
}
```

Si l'API externe est disponible dans l'environnement de l'utilisateur, Flash Player appelle la méthode JavaScript `JS_getBrowserObjects()`, qui passe en boucle sur l'objet navigator du navigateur et renvoie une chaîne de valeurs au format URL à ActionScript. Cette chaîne est alors convertie en un objet URLVariables (`itemVars`) et ajoutée au tableau `itemArr`, qui est renvoyé au script appelant.

Communication avec JavaScript

La dernière étape dans la construction de l'application CapabilitiesExplorer consiste à écrire le code JavaScript nécessaire au passage en boucle de chaque élément de l'objet navigator du navigateur et à l'ajout d'une paire nom-valeur dans un tableau temporaire. Le code de la méthode JavaScript `JS_getBrowserObjects()` dans le fichier container.html est le suivant :

```
<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>
```

Le code commence par créer un tableau temporaire qui contiendra toutes les paires nom-valeur de l'objet navigator. Une boucle `for..in` est ensuite appliquée à l'objet navigator, et le type de données de la valeur actuelle est évaluée pour éliminer les valeurs indésirables. Dans cette application, seules les valeurs String ou Boolean nous intéressent. Les autres types de données (par exemple les fonctions ou les tableaux) sont ignorés. Chaque valeur String ou Boolean de l'objet navigator est ajoutée au tableau `tempArr`. Une boucle `for..in` est ensuite appliquée à l'objet screen du navigateur, et chaque valeur numérique est ajoutée au tableau `tempArr`. Enfin, le tableau temporaire est converti en chaîne à l'aide de la méthode `Array.join()`. Ce tableau utilise l'esperluette (&) comme délimiteur, ce qui permet à ActionScript d'analyser facilement les données à l'aide de la classe URLVariables.

Chapitre 29 : Copie et collage

Utilisez les classes dans le presse-papiers de l'interface de programmation pour copier les informations à destination et en provenance du presse-papiers du système. Les formats des données qui peuvent être transférées en provenance ou à destination d'une application qui s'exécute dans Adobe® AIR™ et Adobe® Flash® Player sont les suivants :

- Bitmaps (AIR uniquement)
- Fichiers (AIR uniquement)
- Texte
- Texte au format HTML
- Données RTF
- Chaînes URL (AIR uniquement)
- Objets sérialisés
- Références d'objet (valides uniquement dans l'application d'origine)

Principes de base de la copie et du collage

L'interface de programmation copie et collage contient les classes suivantes :

Package	Classes
flash.desktop	• presse-papiers • ClipboardFormats • ClipboardTransferMode

La propriété statique `Clipboard.generalClipboard` représente le presse-papiers du système d'exploitation. La classe Clipboard fournit deux méthodes pour la lecture et l'écriture des données dans les objets du presse-papiers.

Les classes HTMLLoader (dans AIR) et TextField implémentent un comportement par défaut pour les raccourcis clavier de copie et collage. Pour implémenter un comportement de raccourci copie et collage pour des composants personnalisés, vous pouvez vous mettre directement à l'écoute de ces frappes. Vous pouvez également utiliser des commandes de menu natives accompagnées d'équivalents de touches pour répondre indirectement aux frappes.

Un objet Clipboard unique peut servir à rendre disponibles des représentations diverses des mêmes informations afin d'augmenter les possibilités des autres applications à comprendre et à utiliser ces données. Par exemple, une image pourrait être incluse en tant que données d'image, un objet Bitmap sérialisé et un fichier. Le rendu des données dans un format peut être reporté de telle sorte que celui-ci ne soit pas effectivement créé tant que les données de ce format ne sont pas lues.

Lecture en provenance et écriture à destination du presse-papiers du système

Pour lire le contenu du presse-papiers du système, appelez la méthode `getData` de l'objet `Clipboard.generalClipbooard` en lui communiquant le nom du format à lire :

```
import flash.desktop.Clipboard;
import flash.desktop.ClipboardFormats;

if(Clipboard.generalClipboard.hasFormat(ClipboardFormats.TEXT_FORMAT)){
    var text:String = Clipboard.generalClipboard.getData(ClipboardFormats.TEXT_FORMAT);
}
```

Remarque : *un contenu qui s'exécute dans Flash Player ou dans un sandbox non applicatif d'AIR ne peut appeler que la méthode `getData()` dans un gestionnaire d'événement pour un événement `paste`. C'est uniquement du code en cours d'exécution dans un sandbox d'application AIR qui peut appeler la méthode `getData()` hors d'un gestionnaire d'événement `Coller`.*

Pour écrire dans le presse-papiers, ajoutez les données à l'objet `Clipboard.generalClipbooard` dans un ou plusieurs formats. Toute donnée existante du même format est automatiquement écrasée. Toutefois, vider le presse-papiers du système avant de lui envoyer de nouvelles données constitue une bonne habitude. On s'assure ainsi que des données sans rapport et dans d'autres formats sont également supprimées.

```
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);
```

Remarque : *un contenu qui s'exécute dans Flash Player ou dans un sandbox non-applicatif d'AIR ne peut appeler que la méthode `setData()` dans un gestionnaire d'événement pour un événement utilisateur, comme un événement de clavier ou de souris, ou bien encore un événement `copy` ou `cut`. C'est uniquement du code en cours d'exécution dans un sandbox d'application AIR qui peut appeler la méthode `setData()` hors d'un gestionnaire d'événement utilisateur.*

Formats de données Clipboard

Les formats de Clipboard décrivent les données placées dans un objet Clipboard. Flash Player ou AIR traduit automatiquement les formats de données standard entre types de données ActionScript et formats de presse-papiers système. De surcroît, les objets application peuvent être transférés au sein des applications basées sur ActionScript et entre celles-ci à l'aide des formats définis par les applications.

Un objet Clipboard peut contenir des représentations des mêmes informations dans différents formats. Par exemple, un objet Clipboard représentant un sprite pourrait inclure un format de référence pour un usage au sein de la même application, un format sérialisé pour un usage par une autre application qui s'exécuterait dans Flash Player ou AIR, un format bitmap pour un usage par un éditeur d'images et un format liste de fichiers (peut-être avec un rendu différé pour coder un fichier PNG) pour permettre une copie ou un glissement d'une représentation du sprite vers le système de fichiers.

Formats de données standard

Les constantes qui définissent les noms des formats standard sont fournis par la classe ClipboardFormats :

Constante	Description
TEXT_FORMAT	Les données au format texte sont transposées en provenance et à destination de la classe String dans ActionScript.
HTML_FORMAT	Texte avec balisage HTML.
RICH_TEXT_FORMAT	Les données au format RTF sont transposées en provenance et à destination de la classe ByteArray d'ActionScript. Le balisage RTF n'est ni interprété ni transposé de quelque façon que ce soit.
BITMAP_FORMAT	(AIR uniquement) Les données au format bitmap sont transposées en provenance et à destination de la classe BitmapData d'ActionScript.
FILE_LIST_FORMAT	(AIR uniquement) Les données au format liste de fichiers sont transposées en provenance et à destination d'un tableau des objets File d'ActionScript.
URL_FORMAT	(AIR uniquement) Les données au format URL sont transposées en provenance et à destination de la classe String d'ActionScript.

Formats de données personnalisés

Vous pouvez utiliser des formats personnalisés définis dans les applications pour transférer des objets en tant que références ou des copies numérotées. Les références ne sont valides qu'au sein d'une même application s'exécutant dans AIR ou Flash Player. Les objets sérialisés peuvent être transférés entre applications s'exécutant dans AIR ou Flash Player, mais ne peuvent être utilisés qu'avec des objets qui demeurent valides lorsqu'ils sont sérialisés et désérialisés. Les objets peuvent généralement être sérialisés si leurs propriétés sont soit des types simples, soit des objets sérialisables.

Pour ajouter un objet sérialisé à un objet Clipboard, définissez le paramètre sérialisable sur `true` lorsque vous appelez la méthode `Clipboard.setData()`. Le nom du format peut être celui d'un format standard ou une chaîne arbitraire définie par votre application.

Modes de transfert

Lorsqu'un objet est écrit sur le presse-papiers à l'aide d'un format de données personnalisé, les données de l'objet peuvent être lues depuis le presse-papiers soit comme une référence, soit comme une copie sérialisée de l'objet d'origine. AIR définit quatre modes de transfert qui déterminent si les objets sont transférés en tant que références ou en tant que copies sérialisées :

Mode de transfert	Description
ClipboardTransferModes.ORIGINAL_ONLY	Seule une référence est renvoyée. Si aucune référence n'est disponible, une valeur nulle est renvoyée.
ClipboardTransferModes.ORIGINAL_PREFFERED	Une référence est renvoyée, si elle est disponible. Sinon, une copie sérialisée est renvoyée.
ClipboardTransferModes.CLONE_ONLY	Seule une copie sérialisée est renvoyée. S'il n'y a pas de copie sérialisée, une valeur " null " est alors renvoyée.
ClipboardTransferModes.CLONE_PREFFERED	Une copie sérialisée est renvoyée, si elle est disponible. Sinon, une référence est renvoyée.

Lecture et écriture des formats de données personnalisés

Vous pouvez utiliser toute chaîne qui ne commence pas par le préfixe `air:` ou `flash:` pour le paramètre format lorsque vous écrivez un objet dans le presse-papiers. Utilisez la même chaîne que le format pour lire l'objet. Les exemples suivants illustrent la façon de lire et d'écrire des objets dans le presse-papiers.

```
public function createClipboardObject(object:Object):Clipboard{
    var transfer:Clipboard = Clipboard.generalClipboard;
    transfer.setData("object", object, true);
}
```

Pour extraire un objet sérialisé de l'objet presse-papiers (à la suite d'une opération déposer ou coller), utilisez le même nom de format et les modes de transfert `cloneOnly` ou `clonePreferred`.

```
var transfer:Object = clipboard.getData("object", ClipboardTransferMode.CLONE_ONLY);
```

Une référence est toujours ajoutée à l'objet Clipboard. Pour extraire la référence de l'objet presse-papiers (à la suite d'une opération déposer ou coller), utilisez les modes de transfert `originalOnly` ou `originalPreferred` en lieu et place de la copie sérialisée.

```
var transferredObject:Object =
    clipboard.getData("object", ClipboardTransferMode.ORIGINAL_ONLY);
```

Les références ne sont valides que si l'objet Clipboard provient de l'application actuelle qui s'exécute dans AIR ou Flash Player. Utilisez le mode de transfert `originalPreferred` pour accéder à la référence lorsqu'elle devient disponible et le clone sérialisé lorsque la référence ne l'est pas.

Rendu différé

Si la création d'un format de données est coûteuse en ressources informatiques, vous pouvez procéder à un rendu différé en fournissant une fonction qui fournit les données à la demande. La fonction n'est appelée que si le récepteur de l'opération déposer ou coller demande les données dans le format différé.

La fonction de rendu est ajoutée à l'objet Clipboard à l'aide de la méthode`setDataHandler()`. La fonction doit renvoyer les données dans un format approprié. Par exemple, si vous appelez `setDataHandler(ClipboardFormat.TEXT_FORMAT, writeText)`, alors la fonction `writeText()` doit renvoyer une chaîne.

Si un format de données de même type est ajouté à l'objet Clipboard avec la méthode `setData()`, ces données auront priorité sur la version différée (la fonction de rendu n'est jamais appelée). La fonction de rendu peut être appelée ou non de nouveau si on accède une deuxième fois aux mêmes données du presse-papiers.

Remarque : *sous Mac OS X, le rendu différé ne fonctionne qu'avec des formats de données personnalisés. Avec des formats de données standard, la fonction de rendu est appelée immédiatement.*

Collage de texte à l'aide d'une fonction de rendu différé

L'exemple ci-dessous illustre la façon d'implémenter une fonction de rendu différé.

Lorsque l'utilisateur appuie sur le bouton Copier, l'application vide le presse-papiers du système pour s'assurer qu'aucune donnée d'opérations de presse-papiers précédentes ne subsiste. La méthode `setDataHandler()` définit alors la fonction `renderData()` comme rendu du presse-papiers.

Lorsque l'utilisateur choisit la commande Coller du menu contextuel du champ de texte de destination, l'application accède au presse-papiers et définit le texte de destination. Comme le format des données texte du presse-papiers a été défini avec une fonction plutôt qu'une chaîne, le presse-papiers appelle la fonction `renderData()`. Cette fonction `renderData()` renvoie le texte dans le texte source qui, lui, est affecté au texte de destination.

Remarquez que si vous éditez le texte source avant d'appuyer sur le bouton Coller, les modifications se retrouveront dans le texte collé, même lorsque l'édition survient après avoir appuyé sur le bouton Copier. Cette situation existe du fait que la fonction de rendu ne copie pas le texte source tant que l'on n'a pas appuyé sur le bouton Coller. Lorsque vous utilisez le rendu différé dans une véritable application, vous pourriez vouloir stocker ou protéger les données source de manière à éviter ce problème.

```
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;
            }
        }
    }
}
```

Chapitre 30 : Impression

Adobe® Flash® Player et Adobe® AIR™ peuvent communiquer avec l'interface d'impression du système d'exploitation, ce qui vous permet de transmettre des pages au spooler de l'imprimante. Chaque page que Flash Player ou AIR envoie au spooler peut comprendre du contenu visible, dynamique ou invisible pour l'utilisateur à l'écran, y compris des valeurs de base de données et du texte dynamique. Par ailleurs, Flash Player et AIR définissent les propriétés de la classe flash.printing.PrintJob en fonction des paramètres d'impression de l'utilisateur, afin d'assurer le formatage correct des pages.

Ce chapitre étudie en détail les stratégies d'utilisation des méthodes et propriétés de la classe flash.printing.PrintJob en vue de créer une tâche d'impression, de lire les paramètres de l'imprimante utilisée et de les ajuster à la tâche d'impression en fonction des informations renvoyées par Flash Player ou AIR et le système d'exploitation de l'utilisateur.

Principes de base de l'impression

Introduction à l'impression

Dans ActionScript 3.0, vous utilisez la classe PrintJob pour créer des instantanés de contenu d'affichage à convertir en représentation encre-et-papier dans une impression. Dans certains cas, définir le contenu à imprimer revient à le définir pour un affichage à l'écran—vous positionnez et dimensionnez des éléments pour créer la disposition souhaitée. Néanmoins, l'impression a des caractéristiques qui la différencient de la disposition à l'écran. Par exemple, les imprimantes utilisent une résolution différente des écrans d'ordinateur. Le contenu d'un écran d'ordinateur est dynamique et peut changer, alors que le contenu imprimé est statique. Lorsque vous planifiez une impression, vous devez tenir compte des contraintes de taille de page fixe et la possibilité d'imprimer plusieurs pages.

Même si ces différences peuvent sembler évidentes, il est important de les avoir en tête lors de la configuration de l'impression avec ActionScript. Etant donné que l'impression précise dépend d'une combinaison des valeurs que vous spécifiez et des caractéristiques de l'imprimante de l'utilisateur, la classe PrintJob comprend des propriétés qui vous permettent de déterminer les caractéristiques importantes de l'imprimante de l'utilisateur et que vous devrez pendre en considération.

Tâches d'impression courantes

Les tâches d'impression courantes suivantes sont décrites dans ce chapitre :

- Démarrage d'une tâche d'impression
- Ajout de pages à une tâche d'impression
- Détermination de l'annulation d'une tâche d'impression par l'utilisateur
- Spécification de l'utilisation de bitmap ou de rendu vectoriel
- Définition de la taille, de l'échelle et de l'orientation d'une page
- Spécification de la zone imprimable du contenu
- Conversion de la taille écran en taille de page
- Impression de plusieurs tâches d'impression

Concepts importants et terminologie

La liste de référence suivante contient les termes importants utilisés dans ce chapitre :

- Spouleur : partie du système d'exploitation ou du logiciel du pilote d'imprimante qui effectue le suivi des pages en attente d'impression et les envoie à l'imprimante dès qu'elle est disponible.
- Orientation de page : rotation du contenu imprimé par rapport au papier (horizontale - paysage ou verticale - portrait).
- Tâche d'impression : page ou jeu de pages constituant une seule impression.

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester les exemples de code. Un grand nombre des codes fournis dans ce chapitre sont de petites portions de code plutôt que des exemples complets d'impression ou des codes qui vérifient des valeurs. Le test des exemples implique la création d'éléments à imprimer et l'utilisation des codes avec ces éléments. Les deux derniers exemples du chapitre sont des exemples complets d'impression. Ils comprennent le code qui définit le contenu à imprimer ainsi que l'exécution des tâches d'impression.

Pour tester les exemples de code :

1. Créez un nouveau document Flash.
2. Sélectionnez l'image-clé sur l'image 1 du scénario puis ouvrez le panneau Actions.
3. Copiez le code dans le panneau Script.
4. Dans le menu principal, choisissez Contrôle > Tester l'animation pour créer le fichier SWF et tester l'exemple.

Impression d'une page

Vous utilisez une occurrence de la classe PrintJob pour gérer l'impression. Pour imprimer une page de base dans Flash Player ou AIR, vous utilisez quatre instructions à la suite :

- `new PrintJob()` : crée une occurrence de tâche d'impression dotée du nom que vous spécifiez.
- `PrintJob.start()` : initialise le processus d'impression pour le système d'exploitation (en appelant la boîte de dialogue d'impression destinée à l'utilisateur) et définit les propriétés en lecture seule de la tâche d'impression.
- `PrintJob.addPage()` : comprend les détails du contenu de la tâche d'impression, notamment l'objet Sprite (et ses éventuels enfants), la taille de la zone d'impression et le mode d'impression d'image (vectoriel ou bitmap) que l'imprimante doit utiliser. Vous pouvez effectuer plusieurs appels successifs de `addPage()` afin d'imprimer plusieurs sprite sur plusieurs pages.
- `PrintJob.send()` : envoie les pages à l'imprimante du système d'exploitation.

Voici donc un exemple de script simple de tâche d'impression (qui inclut les instructions `package`, `import` et `class` en vue de la compilation) :

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

Remarque : *cet exemple vise à illustrer les éléments de base d'un script de tâche d'impression et ne contient aucun dispositif de gestion des erreurs. Pour construire un script qui réponde convenablement à l'annulation d'une tâche d'impression par l'utilisateur, consultez la section « Utilisation des exceptions et des renvois » à la page 686.*

Si vous devez cependant purger les propriétés d'un objet PrintJob, utilisez la variable PrintJob `null` (par exemple `myPrintJob = null`).

Tâches Flash Player et AIR et impression système

Flash Player et AIR transmettent des pages à l'interface d'impression du système d'exploitation ; il est donc important de comprendre la répartition des tâches gérées par Flash Player et AIR, ainsi que de celles gérées par l'interface d'impression du système d'exploitation. Flash Player et AIR peuvent initialiser une tâche d'impression, lire certains des paramètres de page de l'imprimante, transmettre le contenu d'une tâche d'impression au système d'exploitation et vérifier si l'utilisateur ou le système annule une tâche. D'autres processus, tels que l'affichage de boîtes de dialogue spécifiques à l'imprimante, l'annulation d'une tâche d'impression mise en file d'attente ou l'indication de l'état de l'imprimante, sont supervisés par le système d'exploitation. Si Flash Player et AIR sont capables de réagir en cas de problème lors de l'initialisation ou du formatage d'une tâche d'impression, ils peuvent uniquement établir des rapports sur certaines propriétés ou conditions transmises par l'interface d'impression du système d'exploitation. C'est le rôle du développeur d'élaborer un code susceptible de répondre à ces propriétés ou conditions.

Utilisation des exceptions et des renvois

Avant d'appeler `addPage()` et `send()`, il est recommandé de vérifier que la méthode `PrintJob.start()` renvoie la valeur `true`, au cas où l'utilisateur aurait annulé la tâche d'impression. Une manière simple de vérifier si ces méthodes ont été annulées avant de poursuivre consiste à les intégrer à une instruction `if`, comme suit :

```
if (myPrintJob.start())
{
    // addPage() and send() statements here
}
```

Si `PrintJob.start()` renvoie `true`, c'est-à-dire si l'utilisateur a sélectionné Imprimer (ou Flash Player ou AIR a initialisé une commande d'impression), les méthodes `addPage()` et `send()` peuvent être appelées.

En outre, pour faciliter la gestion du processus d'impression, Flash Player et AIR transmettent désormais des exceptions pour la méthode `PrintJob.addPage()`. Vous pouvez ainsi intercepter les erreurs et fournir des informations et des options à l'utilisateur. Si une méthode `PrintJob.addPage()` échoue, il est également possible d'appeler une autre fonction ou d'arrêter la tâche d'impression en cours. Vous interceptez ces exceptions en intégrant des appels `addPage()` à une instruction `try..catch`, comme illustré par l'exemple suivant. Dans cet exemple, `[params]` constitue un espace réservé pour les paramètres spécifiant le contenu réel à imprimer :

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error,
    }
    myPrintJob.send();
}
```

Une fois la tâche d'impression lancée, vous pouvez ajouter le contenu `PrintJob.addPage()` pour voir s'il génère une exception (par exemple si l'utilisateur a annulé la tâche d'impression). Si c'est le cas, vous pouvez soit ajouter une logique à l'instruction `catch` pour fournir à l'utilisateur (ou Flash Player ou AIR) des informations et des options, soit arrêter la tâche d'impression en cours. Si l'ajout de page réussit, vous pouvez procéder à l'envoi des pages vers l'imprimante à l'aide de `PrintJob.send()`.

Si Flash Player ou AIR éprouve des difficultés à envoyer la tâche d'impression à l'imprimante (par exemple, si l'imprimante est hors ligne), vous pouvez intercepter cette exception également. Vous pouvez alors fournir à l'utilisateur (ou à Flash Player ou à AIR) des informations ou davantage d'options (comme l'affichage d'un message ou le lancement d'une alerte accompagnée d'une animation). Vous pouvez, par exemple, affecter un nouveau texte à un champ texte dans une instruction `if..else`, comme illustré par le code suivant :

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (error:Error)
    {
        // Handle error.
    }
    myPrintJob.send();
}
else
{
    myAlert.text = "Print job canceled";
}
```

Pour disposer d'un exemple qui fonctionne, consultez la section « Exemple : redimensionnement, recadrage et ajustement » à la page 692.

Utilisation des propriétés de page

Lorsque l'utilisateur clique sur OK dans la boîte de dialogue d'impression et que `PrintJob.start()` renvoie `true`, vous pouvez accéder aux propriétés définies par les paramètres de l'imprimante. Il s'agit notamment de la largeur et de la hauteur du papier (`pageHeight` et `pageWidth`) et de l'orientation du contenu sur la feuille. En provenance de l'imprimante, ces paramètres, que Flash Player ou AIR ne contrôle pas, ne peuvent pas être modifiés. Vous pouvez en revanche les utiliser pour disposer le contenu à envoyer à l'imprimante en fonction des paramètres en cours. Pour plus d'informations, consultez la section « Définition de la taille, de l'échelle et de l'orientation » à la page 689.

Définition du rendu vectoriel ou bitmap

Vous avez la possibilité de définir manuellement la tâche d'impression de manière à transmettre chaque page sous forme d'image vectorielle ou bitmap. Dans certains cas, l'impression vectorielle produit un fichier de file d'attente plus réduit et une image de meilleure qualité que l'impression bitmap. Toutefois, si le contenu inclut une image bitmap et que vous souhaitiez préserver la transparence alpha ou tout autre effet de couleur, il est recommandé de choisir l'impression bitmap pour cette page. En outre, une imprimante non-PostScript convertit automatiquement les graphiques vectoriels en images bitmap.

Remarque : *Adobe AIR ne prend pas en charge l'impression vectorielle sous Mac OS.*

L'impression bitmap se spécifie dans le troisième paramètre de `PrintJob.addPage()`, par transmission d'un objet PrintJobOptions dont le paramètre `printAsBitmap` a la valeur `true`, comme suit :

```
var options:PrintJobOptions = new PrintJobOptions();
options.printAsBitmap = true;
myPrintJob.addPage(mySprite, null, options);
```

Si vous ne spécifiez aucune valeur pour le troisième paramètre, la tâche d'impression utilise le paramètre par défaut, soit l'impression vectorielle.

Remarque : *si vous ne souhaitez pas spécifier de valeur pour `printArea` (le deuxième paramètre) mais en définir une pour l'impression bitmap, utilisez la valeur `null` pour `printArea`.*

Temporisation des instructions de tâche d'impression

Contrairement aux versions précédentes, ActionScript 3.0 ne limite pas un objet PrintJob à une image unique. Cependant, comme le système d'exploitation indique l'état de l'impression après que l'utilisateur a cliqué sur le bouton OK dans la boîte de dialogue d'impression, appelez `PrintJob.addPage()` et `PrintJob.send()` dès que possible pour envoyer les pages au spouleur. Si l'accès à l'image contenant l'appel `PrintJob.send()` est soumis à un délai, le processus d'impression est également retardé.

Dans ActionScript 3.0, le délai de script est de 15 secondes. Par conséquent, l'intervalle entre chaque instruction essentielle de la séquence de tâche d'impression ne peut pas dépasser 15 secondes. En d'autres termes, la limite de 15 secondes concerne les intervalles suivants :

- Entre `PrintJob.start()` et la première instruction `PrintJob.addPage()`
- Entre `PrintJob.addPage()` et l'instruction suivante `PrintJob.addPage()`
- Entre la dernière instruction `PrintJob.addPage()` et `PrintJob.send()`

Si l'un des intervalles ci-dessus excède 15 secondes, l'appel suivant de la méthode `PrintJob.start()` pour l'occurrence de PrintJob renvoie `false` et l'appel suivant de la méthode `PrintJob.addPage()` pour l'occurrence de PrintJob entraîne le renvoi d'une exception d'exécution par Flash Player ou AIR.

Définition de la taille, de l'échelle et de l'orientation

La section « Impression d'une page » à la page 685 décrit en détail les étapes d'une tâche d'impression de base, dans laquelle la sortie reproduit directement le sprite spécifié, selon sa taille d'affichage et sa position à l'écran. Néanmoins, les imprimantes utilisent différentes résolutions d'impression et certains de leurs paramètres peuvent altérer l'aspect du sprite imprimé.

Flash Player et AIR peuvent lire les paramètres d'impression du système d'exploitation, mais notez que ces propriétés sont accessibles en lecture seule. En d'autres termes, bien que vous puissiez réagir à leur valeur, il est possible de les définir. Par exemple, il est possible de déterminer le paramètre de format de page de l'imprimante et d'ajuster votre contenu en fonction. Vous pouvez de même identifier les paramètres de marge de l'imprimante ainsi que l'orientation des pages. Pour répondre aux paramètres de l'imprimante, il peut s'avérer nécessaire de spécifier une zone d'impression, d'effectuer un ajustement pour tenir compte de la différence entre la résolution de l'écran et la mesure de points de l'imprimante, ou de faire correspondre le contenu aux paramètres de taille et d'orientation de l'imprimante.

Définition de la zone d'impression à l'aide de rectangle

La méthode `PrintJob.addPage()` vous permet de spécifier la partie du sprite que vous souhaitez imprimer. Le deuxième paramètre, `printArea` prend la forme d'un objet Rectangle. Vous pouvez fournir la valeur de ce paramètre de trois manières :

- Créez un objet Rectangle doté de propriétés spécifiques, puis utilisez-le dans l'appel `addPage()`, comme dans l'exemple suivant :

  ```
  private var rect1:Rectangle = new Rectangle(0, 0, 400, 200);
  myPrintJob.addPage(sheet, rect1);
  ```

- Si vous n'avez pas encore spécifié l'objet Rectangle, vous pouvez le faire dans l'appel lui-même, comme illustré ci-dessous :

  ```
  myPrintJob.addPage(sheet, new Rectangle(0, 0, 100, 100));
  ```

- Si vous prévoyez de fournir des valeurs pour le troisième paramètre de l'appel `addPage()` sans pour autant spécifier un objet Rectangle, utilisez la valeur `null` pour le deuxième paramètre, comme suit :

  ```
  myPrintJob.addPage(sheet, null, options);
  ```

Remarque : *si vous envisagez de spécifier un rectangle pour définir les dimensions d'impression, n'oubliez pas d'importer la classe `flash.display.Rectangle`.*

Comparaison entre les points et les pixels

La largeur et la hauteur d'un rectangle sont exprimées en pixels. Une imprimante utilise les points en tant qu'unités de mesure. Les points ont une taille physique fixe (1/72e de pouce), mais la taille d'un pixel à l'écran varie selon la résolution de ce dernier. De ce fait, le taux de conversion entre les pixels et les points dépend de la configuration de l'imprimante et du redimensionnement éventuel du sprite. Un sprite non redimensionné d'une largeur de 72 pixels mesure un pouce (2,54 cm) de large lorsqu'il est imprimé, sachant qu'un point correspond à un pixel quelle que soit la résolution de l'écran.

Vous pouvez utiliser les équivalences suivantes pour convertir les pouces ou les centimètres en twips ou points (un twip correspond à 1/20ème de point) :

- 1 point = 1/72ème de pouce = 20 twips
- 1 pouce = 72 points = 1 440 twips

- 1 centimètre = 567 twips

Si vous omettez le paramètre `printArea` ou s'il est transmis de façon incorrecte, la zone entière du sprite est imprimée.

Redimensionnement

Si vous souhaitez redimensionner un objet Sprite avant de l'imprimer, définissez les propriétés de redimensionnement (voir « Manipulation de la taille et de l'échelle des objets » à la page 305) avant d'appeler la méthode `PrintJob.addPage()`, puis rétablissez leurs valeurs d'origine après l'impression. L'échelle d'un objet Sprite ne dépend pas de la propriété `printArea`. En d'autres termes, si vous spécifiez une zone d'impression de 50 pixels par 50 pixels, 2 500 pixels sont imprimés. Si vous redimensionnez l'objet Sprite, les 2 500 pixels sont imprimés, mais l'objet est imprimé à l'échelle retenue.

A titre d'exemple, consultez la section « Exemple : redimensionnement, recadrage et ajustement » à la page 692.

Impression en mode paysage ou portrait

Flash Player et AIR étant capables de détecter les paramètres d'orientation, vous pouvez insérer dans votre code ActionScript une logique permettant d'ajuster la taille du contenu ou son orientation en fonction des paramètres de l'imprimante, comme illustré dans l'exemple ci-après.

```
if (myPrintJob.orientation == PrintJobOrientation.LANDSCAPE)
{
    mySprite.rotation = 90;
}
```

Remarque : *si vous envisagez de lire les paramètres système pour définir l'orientation du contenu sur le papier, n'oubliez pas d'importer la classe PrintJobOrientation. La classe PrintJobOrientation fournit des valeurs constantes qui définissent l'orientation du contenu sur la page. Vous importez la classe à l'aide de l'instruction suivante :*

```
import flash.printing.PrintJobOrientation;
```

Ajustement de la hauteur et de la largeur au format du papier

Par l'utilisation d'une stratégie semblable à la gestion des paramètres d'orientation de l'imprimante, vous pouvez lire les paramètres de hauteur et de largeur de page, puis en tenir compte en intégrant une logique dans une instruction `if`. Le code suivant illustre ce cas de figure :

```
if (mySprite.height > myPrintJob.pageHeight)
{
    mySprite.scaleY = .75;
}
```

En outre, il est possible de déterminer les paramètres de marge d'une page en comparant les dimensions de cette page à celle du papier, comme illustré ci-après :

```
margin_height = (myPrintJob.paperHeight - myPrintJob.pageHeight) / 2;
margin_width = (myPrintJob.paperWidth - myPrintJob.pageWidth) / 2;
```

Exemple : impression de plusieurs pages

Si vous devez imprimer plusieurs pages de contenu, vous pouvez associer chaque page à un sprite différent (dans le cas présent, `sheet1` et `sheet2`). Utilisez ensuite `PrintJob.addPage()` pour chaque sprite. Le code suivant illustre cette technique :

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

Exemple : redimensionnement, recadrage et ajustement

Dans certains cas, il peut s'avérer utile d'ajuster la taille (ou d'autres propriétés) d'un objet d'affichage à imprimer afin de tenir compte des différences entre son aspect à l'écran et le rendu sur papier. Lorsque vous modifiez les propriétés d'un objet d'affichage avant l'impression (par exemple à l'aide des propriétés `scaleX` et `scaleY`), n'oubliez pas que l'objet redimensionné reste plus grand que le rectangle défini comme zone d'impression, l'objet subira un recadrage. Il est également judicieux d'envisager de rétablir les propriétés après l'impression des pages.

Le code suivant modifie les dimensions de l'objet d'affichage `txt` (sans altérer la zone d'arrière-plan verte). Pour finir, le champ de texte est recadré en fonction des dimensions du rectangle spécifié. Après l'impression, le champ de texte reprend sa taille originale d'affichage à l'écran. Si l'utilisateur annule la tâche d'impression dans la boîte de dialogue d'impression du système d'exploitation, le contenu de Flash Player ou AIR est modifié pour avertir l'utilisateur de cette annulation.

```
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;
        }
    }
}
```

Chapitre 31 : Utilisation de l'API externe

L'API externe ActionScript 3.0 permet une communication simple entre ActionScript et l'application conteneur dans laquelle Adobe Flash Player s'exécute. L'API externe peut vous être utile dans plusieurs cas de figure, par exemple si vous créez une interaction entre un document SWF et JavaScript dans une page HTML ou si vous construisez une application de bureau qui affiche un fichier SWF à l'aide de Flash Player.

Ce chapitre décrit comment utiliser l'API externe pour interagir avec une application conteneur, comment transmettre des données entre ActionScript et JavaScript dans une page HTML et comment établir une communication et échanger des données entre ActionScript et une application de bureau.

Remarque : *ce chapitre traite uniquement de la communication entre ActionScript dans un fichier SWF et l'application qui contient une référence à Flash Player ou à l'occurrence dans laquelle ce fichier SWF est chargé. Toute autre utilisation de Flash Player dans une application n'est pas traitée dans cette documentation. Flash Player est conçu pour être utilisé comme plug-in de navigation ou de projection (application autonome). Les autres scénarios d'utilisation peuvent avoir une prise en charge limitée.*

Principes de base de l'utilisation de l'API externe

Introduction à l'utilisation de l'API externe

Même si dans certains cas, un fichier SWF peut s'exécuter tout seul (par exemple, si vous créez un fichier de projection SWF), généralement, une application SWF s'exécute comme un élément intégré dans une autre application. De façon générale, le conteneur dans lequel le fichier SWF est intégré est un fichier HTML ; un fichier SWF est utilisé, moins fréquemment, pour une partie ou l'intégralité de l'interface utilisateur d'une application de bureau.

Au fur et à mesure que vous utilisez des applications plus avancées, il se peut que vous souhaitiez définir une communication entre le fichier SWF et l'application du conteneur. Par exemple, il est courant pour une page Web d'afficher du texte ou d'autres informations en HTML, et d'inclure un fichier SWF pour afficher du contenu visuel dynamique (diagramme ou vidéo). Dans ce cas, vous souhaitez peut-être que lorsque les utilisateurs cliquent sur un bouton de la page Web, le fichier SWF soit modifié. ActionScript contient un mécanisme connu sous le nom d'API externe, qui facilite ce type de communication entre ActionScript dans un fichier SWF et d'autre code dans l'application conteneur.

Tâches d'API externe courantes

Les tâches d'API externe courantes suivantes sont décrites dans ce chapitre :

- Obtention d'informations sur l'application conteneur
- Utilisation d'ActionScript pour appeler un code dans une application conteneur, y compris une page Web ou une application de bureau
- Appel de code ActionScript depuis un code dans une application conteneur
- Création d'un proxy pour simplifier l'appel de code ActionScript depuis une application conteneur

Concepts importants et terminologie

La liste de référence suivante contient les termes importants utilisés dans ce chapitre :

- Conteneur Active X : une application conteneur (pas un navigateur Web) qui comprend une occurrence du contrôle ActiveX Flash Player pour afficher le contenu SWF dans l'application.
- Application conteneur : l'application dans laquelle Flash Player exécute un fichier SWF (un navigateur Web et une page HTML qui comprend un contenu Flash Player, par exemple).
- Projection : un fichier SWF qui a été converti en un fichier exécutable autonome comprenant Flash Player et le contenu du fichier SWF. Un fichier de projection peut être créé dans Adobe Flash CS4 Professional ou à l'aide du Flash Player autonome. Les fichiers de projection sont généralement utilisés pour distribuer des fichiers SWF par CD-ROM ou dans des situations semblables, lorsque le volume à télécharger n'est pas un problème et que l'auteur du SWF souhaite être sûr que l'utilisateur pourra exécuter le fichier SWF, indépendamment du fait que Flash Player est installé sur l'ordinateur de l'utilisateur.
- Proxy : un code ou une application intermédiaire qui appelle du code dans une application (l'application externe) au nom d'une autre application (l'application appelante), et renvoie des valeurs à l'application appelante. Un proxy peut être utilisé pour différentes raisons, notamment :
  - Pour simplifier le processus d'exécution d'appels de fonction externe en convertissant des appels de fonction native dans l'application appelante au format compris par l'application externe
  - Pour contourner les limites de sécurité ou autres qui empêchent l'appelant de communiquer directement avec l'application externe
- Sérialiser : convertir des objets ou des valeurs de données en un format qui peut être utilisé pour transmettre les valeurs dans des messages entre deux systèmes de programmation (par exemple, sur Internet ou entre deux applications différentes s'exécutant sur un seul ordinateur).

Utilisation des exemples fournis dans ce chapitre

Au fur et à mesure que vous avancez dans ce chapitre, vous pouvez tester les exemples de code. Un grand nombre des codes fournis dans ce chapitre sont de petite taille à des fins de démonstration, plutôt que des exemples complets ou des codes qui vérifient des valeurs. Etant donné que l'utilisation de l'API externe exige (par définition) l'écriture de code ActionScript ainsi que de code dans une application conteneur, le test des exemples implique la création d'un conteneur (par exemple, une page Web contenant le SWF) et l'utilisation des codes pour interagir avec le conteneur.

Pour tester un exemple de communication entre ActionScript et JavaScript :

1. Créez un document vide à l'aide de l'outil de programmation Flash et enregistrez-le dans votre ordinateur.
2. Dans le menu principal, choisissez Fichier > Paramètres de publication.
3. Dans l'onglet Formats de la boîte de dialogue Paramètres de publication, vérifiez que les cases à cocher Flash et HTML sont sélectionnées.
4. Cliquez sur le bouton Publier. Ceci génère un fichier SWF et un fichier HTML dans le même dossier et avec le même nom que vous avez utilisé pour enregistrer le document Fermez la boîte de dialogue Paramètres de publication en cliquant sur le bouton OK.
5. Désélectionnez la case à cocher HTML. Une fois que la page HTML est générée, vous pouvez la modifier pour ajouter le code JavaScript approprié. Lorsque vous désactivez l'option HTML, vous vous assurez qu'après avoir modifié la page HTML, Flash n'écrasera pas vos changements avec une nouvelle page HTML lors de la publication du fichier SWF.
6. Fermez la boîte de dialogue Paramètres de publication en cliquant sur le bouton OK.

7 Avec une application d'éditeur de texte ou HTML, ouvrez le fichier HTML créé par Flash lorsque vous avez publié le SWF. Dans le code source HTML, ajoutez des balises `script` d'ouverture et de fermeture et copiez-y le code JavaScript issu de l'exemple de code :

```
<script>
// add the sample JavaScript code here
</script>
```

8 Enregistrez le fichier HTML et revenez à Flash.

9 Sélectionnez l'image-clé sur l'image 1 du scénario puis ouvrez le panneau Actions.

10 Copiez le code ActionScript dans le panneau Script.

11 Dans le menu principal, choisissez Fichier > Publier pour mettre à jour le fichier SWF avec les changements que vous avez effectués.

12 Utilisez un navigateur Web pour ouvrir la page HTML que vous avez modifiée afin d'afficher la page et de tester la communication entre ActionScript et la page HTML.

Pour tester un exemple de communication de conteneur ActionScript-vers-ActiveX :

1 Créez un document vide à l'aide de l'outil de programmation Flash et enregistrez-le dans votre ordinateur. Vous pouvez l'enregistrer dans le dossier dans lequel votre application conteneur s'attend à trouver le fichier SWF.

2 Dans le menu principal, choisissez Fichier > Paramètres de publication.

3 Dans l'onglet Formats de la boîte de dialogue Paramètres de publication, vérifiez que seule la case à cocher Flash est sélectionnée.

4 Dans le champ Fichier situé en regard de la case à cocher Flash, cliquez sur l'icône de dossier pour sélectionner le dossier dans lequel votre fichier SWF sera publié. Définissez l'emplacement de votre fichier SWF pour pouvoir (par exemple) conserver le document de programmation dans un dossier, mais placer le fichier SWF publié dans un autre (le dossier contenant le code source pour l'application conteneur, par exemple).

5 Sélectionnez l'image-clé sur l'image 1 du scénario puis ouvrez le panneau Actions.

6 Copiez le code ActionScript pour l'exemple dans le panneau Script.

7 Dans le menu principal, choisissez Fichier > Publier pour publier de nouveau le fichier SWF.

8 Créez et exécutez votre application conteneur afin de tester la communication entre elle et ActionScript.

Les deux exemples fournis à la fin de ce chapitre sont des exemples complets d'utilisation de l'API externe pour communiquer avec une page HTML et une application de bureau C#, respectivement. Ces exemples comprennent le code entier (y compris le code ActionScript et de vérification des erreurs du conteneur) que vous devez utiliser lorsque vous écrivez le code à l'aide de l'API externe. Pour consulter un autre exemple complet d'utilisation de l'API externe, consultez l'exemple de classe pour la classe ExternalInterface dans le Guide de référence du langage et des composants ActionScript 3.0.

Avantages de l'API externe et conditions requises

L'API externe correspond à la partie d'ActionScript qui fournit le mécanisme de communication entre ActionScript et le code exécuté dans une application dite externe, c'est-à-dire qui joue le rôle de conteneur pour Flash Player (en général un navigateur Web ou une application de projection autonome). Dans ActionScript 3.0, la fonctionnalité de l'API externe est assurée par la classe ExternalInterface. Dans les versions de Flash Player antérieures à Flash Player 8, l'action `fscommand()` était utilisée pour établir les communications avec l'application conteneur. La classe ExternalInterface vient remplacer `fscommand()` ; son utilisation est recommandée pour toutes les communications entre JavaScript et ActionScript.

Remarque : *si vous devez utiliser l'ancienne fonction `fscommand()` (par exemple pour maintenir la compatibilité avec d'anciennes applications ou pour interagir avec une application conteneur SWF tierce ou avec le lecteur autonome Flash Player), elle est disponible au niveau du package flash.system.*

La classe ExternalInterface est un sous-système qui simplifie les communications d'ActionScript et Flash Player avec JavaScript dans une page HTML ou avec toute application de bureau qui incorpore une occurrence de Flash Player.

La classe ExternalInterface est disponible uniquement dans les circonstances suivantes :

- Dans toutes les versions prises en charge d'Internet Explorer pour Windows (5.0 et ultérieure)
- Dans une application conteneur telle qu'une application de bureau utilisant une occurrence du contrôle ActiveX Flash Player
- Dans un navigateur qui prend en charge l'interface NPRuntime, qui actuellement comprend Firefox 1.0 et versions ultérieures, Mozilla 1.7.5 et versions ultérieures, Netscape 8.0 et versions ultérieures, ainsi que Safari 1.3 et versions ultérieures.

Dans tous les autres cas de figure (par exemple exécution dans un lecteur autonome), la propriété `ExternalInterface.available` renvoie la valeur `false`.

Depuis ActionScript, vous pouvez appeler une fonction JavaScript sur la page HTML. L'API externe apporte les améliorations fonctionnelles suivantes grâce à `fscommand()` :

- Vous pouvez utiliser toutes les fonctions JavaScript, pas seulement les fonctions compatibles avec `fscommand()`.
- Vous pouvez transmettre n'importe quel nombre d'arguments, avec n'importe quels noms ; vous n'êtes pas limité à une commande et une chaîne d'argument. L'API externe offre donc bien plus de souplesse que `fscommand()`.
- Vous pouvez transmettre divers types de données (Boolean, Number et String, entre autres) ; vous n'êtes plus limité aux paramètres String.
- Vous pouvez recevoir la valeur d'un appel, et cette valeur retourne immédiatement à ActionScript (comme la valeur de retour d'un appel que vous faites).

Important : *si le nom donné à l'occurrence de Flash Player dans une page HTML (l'attribut `id` de la balise `object`) inclut un trait d'union (`-`) ou d'autres caractères définis comme des opérateurs dans JavaScript (par exemple, `+`, `*`, `/`, `\`, `.`, etc.), les appels ExternalInterface issus d'ActionScript ne fonctionneront pas lorsque la page Web de conteneur est affichée dans Internet Explorer.Les appels ExternalInterface ne fonctionnent pas non plus si les balises HTML qui définissent l'occurrence de Flash Player (`object` et `embed`) sont imbriquées dans une balise HTML `form`.*

Utilisation de la classe ExternalInterface

La communication entre ActionScript et l'application conteneur peut se faire de deux façons : soit ActionScript appelle un élément de code (par exemple une fonction JavaScript) défini dans le conteneur, soit le code du conteneur appelle une fonction ActionScript définie comme pouvant être appelée. Dans les deux cas, des informations peuvent être envoyées au code appelé et les résultats renvoyés au code appelant.

Pour faciliter la communication, la classe ExternalInterface comprend deux propriétés statiques et deux méthodes statiques également. Ces propriétés et méthodes servent à obtenir des informations sur la connexion à l'interface externe, à exécuter le code dans le conteneur à partir d'ActionScript et de rendre disponibles les fonctions ActionScript que le conteneur doit appeler.

Obtention d'informations sur le conteneur externe

La propriété `ExternalInterface.available` indique si Flash Player se trouve dans un conteneur offrant une interface externe. Si l'interface externe est disponible, cette propriété est `true` ; sinon, elle est `false`. Avant d'utiliser toute autre fonctionnalité de la classe ExternalInterface, vous devez toujours vérifier que le conteneur actif prend en charge la communication avec l'interface externe, comme suit :

```
if (ExternalInterface.available)
{
    // Perform ExternalInterface method calls here.
}
```

Remarque : *la propriété `ExternalInterface.available` indique si le conteneur actif prend en charge la connectivité ExternalInterface. Elle ne vous dit pas si JavaScript est activé dans le navigateur actif.*

La propriété `ExternalInterface.objectID` vous permet de déterminer l'identifiant unique de l'occurrence Flash Player (c'est-à-dire, l'attribut `id` de la balise `object` dans Internet Explorer ou l'attribut `name` de la balise `embed` dans les navigateurs utilisant l'interface NPRuntime). Cet identifiant unique représente le document SWF actif dans le navigateur et permet d'y faire référence, par exemple si vous appelez une fonction JavaScript dans une page HTML conteneur. Si le conteneur Flash Player n'est pas un navigateur Web, la propriété est `null`.

Appel de code externe à partir d'ActionScript

La méthode `ExternalInterface.call()` exécute le code dans l'application conteneur. Elle nécessite au moins un paramètre, une chaîne contenant le nom de la fonction à appeler dans cette application. Tout paramètre supplémentaire transmis à la méthode `ExternalInterface.call()` est retransmis au conteneur comme paramètres de l'appel de fonction.

```
// 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);
```

Si le conteneur est une page HTML, la méthode appelle la fonction JavaScript avec le nom spécifié, qui doit être défini dans un élément `script` de la page HTML. La valeur de retour de la fonction JavaScript est retransmise à ActionScript.

```
<script language="JavaScript">
    // adds two numbers, and sends the result back to ActionScript
    function addNumbers(num1, num2)
    {
        return (num1 + num2);
    }
</script>
```

Si le conteneur est un autre conteneur ActiveX, le contrôle ActiveX Flash Player distribue alors son événement `FlashCall`. Flash Player sérialise le nom de fonction spécifié et les éventuels paramètres dans une chaîne XML. Le conteneur peut accéder à ces informations dans la propriété `request` de l'objet événement, puis les utiliser pour déterminer comment il doit exécuter son propre code. Pour renvoyer une valeur à ActionScript, le code conteneur appelle la méthode `SetReturnValue()` de l'objet ActiveX et transmet le résultat (sérialisé dans une chaîne XML) comme paramètre de cette méthode. Pour plus d'informations sur le format XML utilisé pour cette communication, consultez la rubrique « Format XML de l'API externe » à la page 701.

Que le conteneur soit un navigateur Web ou un autre conteneur ActiveX, si l'appel échoue ou que la méthode conteneur ne spécifie aucune valeur de retour, la valeur `null` est alors renvoyée. La méthode `ExternalInterface.call()` renvoie une exception SecurityError si l'environnement conteneur appartient à un sandbox de sécurité auquel le code appelant n'a pas accès. Vous pouvez contourner ce problème en attribuant à `allowScriptAccess` une valeur adaptée dans l'environnement conteneur. Par exemple, vous changez la valeur de `allowScriptAccess` dans une page HTML, vous devez modifier l'attribut approprié dans les balises `object` et `embed`.

Appel du code ActionScript à partir du conteneur

Un conteneur peut uniquement appeler du code ActionScript compris dans une fonction, et aucun autre code ActionScript. Pour appeler une fonction ActionScript à partir de l'application conteneur, deux opérations sont nécessaires : enregistrer la fonction auprès de la classe ExternalInterface, puis l'appeler à partir du code du conteneur.

Dans un premier temps, vous devez enregistrer votre fonction ActionScript pour indiquer qu'elle doit être mise à disposition du conteneur. Pour ce faire, utilisez la méthode `ExternalInterface.addCallback()` comme suit :

```
function callMe(name:String):String
{
    return "busy signal";
}
ExternalInterface.addCallback("myFunction", callMe);
```

La méthode `addCallback()` prend deux paramètres : le premier, un nom de fonction sous forme de chaîne, correspond au nom de la fonction pour le conteneur. Le second paramètre est la fonction ActionScript elle-même, qui doit s'exécuter lorsque le conteneur appelle le nom de fonction défini. Comme ces noms sont différents, le nom que le conteneur utilise peut être différent du véritable nom de la fonction ActionScript. Cela vous servira particulièrement si vous ne connaissez pas le nom de la fonction, par exemple si une fonction anonyme est spécifiée ou si la fonction à appeler est déterminée au moment de l'exécution.

Une fois que la fonction ActionScript a été enregistrée auprès de la classe ExternalInterface, le conteneur peut alors appeler la fonction. La procédure d'appel varie en fonction du type de conteneur. Par exemple, si le conteneur est du code JavaScript dans un navigateur, la fonction ActionScript est appelée avec le nom de fonction enregistré, comme s'il s'agissait d'une méthode de l'objet de navigateur Flash Player (c'est-à-dire une méthode de l'objet JavaScript représentant la balise `object` ou `embed`). En d'autres termes, les paramètres sont transmis et le résultat est renvoyé comme si une fonction locale était appelée.

```
<script language="JavaScript">
    // callResult gets the value "busy signal"
    var callResult = flashObject.myFunction("my name");
</script>
...
<object id="flashObject"...>
    ...
    <embed name="flashObject".../>
</object>
```

Si vous appelez une fonction ActionScript dans un fichier SWF exécuté dans une application de bureau, le nom de fonction enregistré et les éventuels paramètres doivent être sérialisés dans une chaîne au format XML. L'appel s'effectue alors sur la méthode `CallFunction()` du contrôle ActiveX avec comme paramètre la chaîne XML obtenue. Pour plus d'informations sur le format XML utilisé pour cette communication, consultez la rubrique « Format XML de l'API externe » à la page 701.

Dans les deux cas, la valeur de retour de la fonction ActionScript est retransmise au code du conteneur, directement sous forme de valeur si l'appelant est un code JavaScript dans un navigateur ou sous forme de chaîne XML s'il s'agit d'un conteneur ActiveX.

Format XML de l'API externe

La communication entre ActionScript et une application hébergeant le contrôle Active X Shockwave Flash nécessite un format XML spécifique qui servira à convertir les appels de fonction et les valeurs. Le format XML utilisé par l'API externe se divise en deux parties. L'une est destinée à représenter les appels de fonction. L'autre sert à représenter des valeurs individuelles ; ce format s'applique aux paramètres des fonctions comme aux valeurs de retour. Le format XML des appels de fonction est utilisé pour les appels en provenance et à destination d'ActionScript. Pour un appel de fonction issu d'ActionScript, Flash Player transmet les informations XML au conteneur ; pour un appel provenant du conteneur, Flash Player attend de l'application une chaîne XML du même format. L'extrait XML suivant donne un exemple d'appel de fonction formaté :

```
<invoke name="functionName" returntype="xml">
    <arguments>
        ... (individual argument values)
    </arguments>
</invoke>
```

Le nœud racine est le nœud `invoke`. Il possède deux attributs : `name` indique le nom de la fonction à appeler et `returntype` a toujours la valeur `xml`. Si l'appel de fonction inclut des paramètres, le nœud `invoke` possède un nœud enfant `arguments`, dont les enfants seront les valeurs de paramètres formatées à l'aide du format de valeur individuelle expliqué ci-après.

Les valeurs individuelles, notamment les paramètres de fonction et les valeurs de retour, utilisent un format qui comprend des informations sur le type de données, en plus des valeurs elles-mêmes : Le tableau suivant répertorie les classes ActionScript et le format XML utilisé pour coder des valeurs de ce type de données :

Classe/valeur ActionScript	Classe/valeur C#	Format	Commentaires
null	null	<null/>	
Boolean `true`	bool true	<true/>	
Boolean `false`	bool false	<false/>	
String	string	<string>valeur de chaîne</string>	

Classe/valeur ActionScript	Classe/valeur C#	Format	Commentaires
Number, int, uint	single, double, int, uint	`<number>27.5</number>` `<number>-12</number>`	

Array (combinaison possible de divers types d'éléments)	Une collection qui accepte des éléments de plusieurs types, ex. ArrayList ou object[]	`<array>`
		`<property id="0">`
		`<number>27.5</number>`
		`</property>`
		`<property id="1">`
		`<string>Hello there!</string>`
		`</property>`
		`...`
		`</array>`
			Le nœud `property` définit des éléments individuels et l'attribut `id` est l'index numérique en base zéro.

Objet	Un dictionnaire contenant des clés de chaîne et des valeurs d'objet, ex. HashTable avec clés de chaînes	`<object>`
		`<property id="name">`
		`<string>John Doe</string>`
		`</property>`
		`<property id="age">`
		`<string>33</string>`
		`</property>`
		`...`
		`</object>`
			Le nœud `property` définit des propriétés individuelles et l'attribut `id` est le nom de la propriété (une chaîne).

Autres classe intégrées ou personnalisées		`<null/> or` `<object></object>`	ActionScript convertit les autres objets en valeur null ou en objet vide. Dans les deux cas, toutes les valeurs de propriété sont perdues.

Remarque : *à titre d'exemple, ce tableau indique des classes C# équivalentes en plus des classes ActionScript. Néanmoins, l'API externe peut être utilisée pour communiquer avec un langage ou un moteur d'exécution prenant en charge les contrôles ActiveX, et n'est pas limitée aux applications C#.*

Lorsque vous créez vos propres applications à l'aide de l'API externe avec une application conteneur ActiveX, il est commode d'écrire un proxy qui effectuera la tâche de conversion des appels de fonction native au format XML sérialisé. Pour consulter un exemple de classe proxy écrite dans C#, consultez la section « Fonctionnement interne de la classe ExternalInterfaceProxy » à la page 712.

Exemple : utilisation de l'API externe dans un conteneur de page Web

Cette application exemple illustre les techniques de communication possibles entre ActionScript et JavaScript au sein d'un navigateur Web. Il s'agit d'une application de messagerie instantanée qui permet à l'utilisateur de discuter avec lui-même (d'où le nom de l'application : Introvert IM). L'API externe permet d'envoyer les messages entre un formulaire HTML dans la page Web et une interface SWF. Les techniques décrites dans cet exemple sont les suivantes :

- Vérification de la disponibilité du navigateur avant d'établir la communication afin de garantir une initialisation correcte
- Vérification de la prise en charge de l'API externe par le conteneur
- Appel des fonctions JavaScript à partir d'ActionScript, transmission des paramètres et réception des valeurs en retour
- Mise à disposition des méthodes ActionScript que JavaScript doit appeler et exécution de ces appels

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers de l'application Introvert IM se trouvent dans le dossier Samples/IntrovertIM_HTML. L'application se compose des fichiers suivants :

Fichier	Description
IntrovertIMApp.fla ou IntrovertIMApp.mxml	Le fichier d'application principal pour Flash (FLA) ou Flex (MXML)
com/example/programmingas3/introvertIM/IMManager.as	Classe établissant et gérant les communications entre ActionScript et le conteneur.
com/example/programmingas3/introvertIM/IMMessageEvent.as	Type d'événement personnalisé distribué par la classe IMManager à la réception d'un message du conteneur.
com/example/programmingas3/introvertIM/IMStatus.as	Enumération dont les valeurs représentent les différents statuts de disponibilité pouvant être sélectionnés dans l'application.
html-flash/IntrovertIMApp.html ou html-template/index.template.html	La page HTML pour l'application pour Flash (html-flash/IntrovertIMApp.html) ou le modèle utilisé pour créer la page HTML du conteneur pour l'application pour Adobe Flex (html-template/index.template.html). Ce fichier contient toutes les fonctions JavaScript qui constitue le conteneur de l'application.

Préparation de la communication entre ActionScript et le navigateur

L'API externe est le plus souvent utilisée pour permettre aux applications ActionScript de communiquer avec le navigateur Web. Grâce à elle, les méthodes ActionScript peuvent appeler du code écrit dans JavaScript, et inversement. En raison de la complexité des navigateurs et de leurs processus internes de rendu des pages, il est impossible de garantir qu'un document SWF pourra enregistrer ses rappels avant l'exécution du premier code JavaScript de la page HTML. Par conséquent, avant d'appeler les fonctions du document SWF à partir de JavaScript, le document SWF doit toujours appeler la page HTML pour lui indiquer qu'il est prêt à accepter des connexions.

Par exemple, grâce à une série d'étapes effectuées par la classe IMManager, Introvert IM détermine si le navigateur est prêt à communiquer et prépare le fichier SWF à la communication. La première étape, qui vérifie que le navigateur est prêt à communiquer, a lieu dans le constructeur IMManager, comme suit :

```
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.");
    }
}
```

Tout d'abord, le code vérifie si l'API externe est disponible dans le conteneur actuel à l'aide de la propriété `ExternalInterface.available`. Si c'est le cas, le processus de mise en place de la communication commence. Pour parer aux éventuelles exceptions de sécurité et autres erreurs qui peuvent se produire pendant les communications avec une application externe, le code est enveloppé dans un bloc `try` (les blocs `catch` correspondants ont été omis de l'exemple pour plus de concision).

Le code appelle ensuite la méthode `isContainerReady()`, présentée ici :

```
private function isContainerReady():Boolean
{
    var result:Boolean = ExternalInterface.call("isReady");
    return result;
}
```

La méthode `isContainerReady()` utilise à son tour la méthode `ExternalInterface.call()` pour appeler la fonction JavaScript `isReady()`, comme suit :

```
<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>
```

La fonction `isReady()` renvoie simplement la valeur de la variable `jsReady`. Cette variable a au départ la valeur `false`. Une fois que l'événement `onload` de la page Web est déclenché, la valeur de la variable devient `true`. En d'autres termes, si ActionScript appelle la fonction `isReady()` avant que la page soit chargée, JavaScript renvoie la valeur `false` à `ExternalInterface.call("isReady")`, à la suite de quoi la méthode ActionScript `isContainerReady()` renvoie la valeur `false`. Une fois la page chargée, la fonction JavaScript `isReady()` renvoie la valeur `true`, donc la méthode ActionScript `isContainerReady()` renvoie aussi la valeur `true`.

Dans le constructeur IMManager, deux solutions sont possibles en fonction de la disponibilité du conteneur. Si `isContainerReady()` renvoie la valeur `true`, le code appelle simplement la méthode `setupCallbacks()`, qui achève la mise en place de la communication avec JavaScript. Dans l'autre cas, si `isContainerReady()` renvoie la valeur `false`, le processus est pratiquement mis en attente. Un objet Timer est créé pour appeler la méthode `timerHandler()` toutes les 100 millisecondes, comme suit :

```
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();
    }
}
```

Chaque fois que la méthode `timerHandler()` est appelée, elle vérifie à nouveau le résultat de la méthode `isContainerReady()`. Lorsque le conteneur est initialisé, la méthode renvoie la valeur `true`. Le code arrête alors le minuteur et appelle la méthode `setupCallbacks()` pour finir la mise en place de la communication avec le navigateur.

Présentation des méthodes ActionScript à JavaScript

Comme le montre l'exemple précédent, une fois que le code établit que le navigateur est prêt, la méthode `setupCallbacks()` est appelée. Cette méthode prépare ActionScript pour recevoir des appels à partir de JavaScript, comme le montre cet exemple :

```
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");
}
```

La méthode `setCallBacks()` achève la préparation de la communication avec le conteneur en appelant `ExternalInterface.addCallback()` afin d'enregistrer deux méthodes qui pourront être appelées par JavaScript. Dans ce code, le premier paramètre (le nom qui sert à désigner la méthode dans JavaScript, soit `"newMessage"` et `"getStatus"`) est identique au nom de la méthode dans ActionScript. (Dans ce cas, il n'y avait pas d'intérêt à utiliser des noms différents, on a donc réutilisé les mêmes noms par souci de simplification.) Enfin, la méthode `ExternalInterface.call()` est utilisée pour appeler la fonction JavaScript `setSWFIsReady()`, qui avertit le conteneur que les fonctions ActionScript ont été enregistrées.

Communication d'ActionScript vers le navigateur

L'application Introvert IM met en évidence plusieurs exemples d'appel de fonctions JavaScript dans la page conteneur. Dans le cas le plus simple (un exemple issu de la méthode `setupCallbacks()`), la fonction JavaScript `setSWFIsReady()` est appelée sans transmettre de paramètres ni recevoir de valeur en retour :

```
ExternalInterface.call("setSWFIsReady");
```

Dans un autre exemple issu de la méthode `isContainerReady()`, ActionScript appelle la fonction `isReady()` et reçoit une valeur booléenne en réponse :

```
var result:Boolean = ExternalInterface.call("isReady");
```

Vous pouvez également transmettre des paramètres aux fonctions JavaScript à l'aide de l'API externe. Considérez par exemple la méthode `sendMessage()` de la classe IMManager, qui est appelée lorsque l'utilisateur envoie un nouveau message à son interlocuteur.

```
public function sendMessage(message:String):void
{
    ExternalInterface.call("newMessage", message);
}
```

Là encore, `ExternalInterface.call()` sert à appeler la fonction JavaScript désignée pour avertir le navigateur du nouveau message. En outre, le message lui-même est transmis comme paramètre supplémentaire à `ExternalInterface.call()`. Il est ensuite transmis comme paramètre à la fonction JavaScript `newMessage()`.

Appel du code ActionScript à partir de JavaScript

Une communication se fait normalement de manière bidirectionnelle, ce que respecte l'application Introvert IM. D'un côté, le client de messagerie Flash Player appelle JavaScript pour envoyer des messages, de l'autre, le formulaire HTML appelle le code JavaScript pour envoyer des messages au fichier SWF et recevoir de celui-ci des informations. Par exemple, lorsque le fichier SWF avertit le conteneur qu'il a établi le contact et peut communiquer, la première action du navigateur consiste à appeler la méthode `getStatus()` de la classe IMManager pour recevoir du client de messagerie SWF le statut de disponibilité de l'utilisateur initial. Cela se fait dans la page Web, avec la fonction `updateStatus()`, comme illustré ci-après :

```
<script language="JavaScript">
...
function updateStatus()
{
    if (swfReady)
    {
        var currentStatus = getSWF("IntrovertIMApp").getStatus();
        document.forms["imForm"].status.value = currentStatus;
    }
}
...
</script>
```

Le code vérifie la valeur de la variable `swfReady`, qui vérifie si le fichier SWF a averti le navigateur qu'il avait enregistré ses méthodes auprès de la classe ExternalInterface. Si le fichier SWF est prêt à recevoir la communication, la ligne suivante (`var currentStatus =` ...) appelle la méthode `getStatus()` dans la classe IMManager. Trois opérations se produisent dans cette ligne de code :

- La fonction JavaScript `getSWF()` est appelée et renvoie une référence à l'objet JavaScript représentant le fichier SWF. Le paramètre transmis à `getSWF()` détermine si l'objet de navigateur est renvoyé, au cas où il y aurait plus d'un fichier SWF dans la page HTML. La valeur transmise à ce paramètre doit correspondre à l'attribut `id` de la balise `object` et à l'attribut `name` de la balise `embed`, toutes deux utilisées pour inclure le fichier SWF.
- Avec la référence au fichier SWF, la méthode `getStatus()` est appelée comme s'il s'agissait d'une méthode de l'objet SWF. Dans ce cas, le nom de fonction « `getStatus` ».est utilisé, car c'est le nom sous lequel la fonction ActionScript a été enregistrée à l'aide de `ExternalInterface.addCallback()`.
- La méthode ActionScript `getStatus()` renvoie une valeur qui est attribuée à la variable `currentStatus`, laquelle devient ensuite le contenu (la valeur `value`) du champ de texte `status`.

Remarque : *si vous vous référez au code, vous avez probablement remarqué que dans le code source relatif à la fonction `updateStatus()`, la ligne qui appelle la fonction `getSWF()` est en fait écrite comme suit : var currentStatus = getSWF("${application}").getStatus(). Le texte `${application}` est un espace réservé dans le modèle de page HTML. Lorsque Adobe Flex Builder 3 génère la page HTML en tant que telle pour l'application, cet espace réservé est remplacé par le texte faisant office d'attribut `id` de la balise `object` et d'attribut `name` de la balise `embed` (soit `IntrovertIMApp` dans l'exemple). Il s'agit de la valeur attendue par la fonction `getSWF()`.*

La fonction JavaScript `sendMessage()` illustre la transmission d'un paramètre à la fonction ActionScript. (`sendMessage()` est la fonction appelée lorsque l'utilisateur appuie sur le bouton Envoyer de la page HTML.)

```
<script language="JavaScript">
...
function sendMessage(message)
{
    if (swfReady)
    {
        ...
        getSWF("IntrovertIMApp").newMessage(message);
    }
}
...
</script>
```

La méthode ActionScript `newMessage()` attend un seul paramètre. De ce fait, la variable JavaScript `message` est transmise à ActionScript en l'utilisant comme paramètre dans l'appel de la méthode `newMessage()` du code JavaScript.

Détection du type de navigateur

L'accès au contenu varie d'un navigateur à l'autre. Pour cette raison, il est important de toujours utiliser JavaScript pour détecter le navigateur utilisé et accéder ensuite à la séquence selon la syntaxe propre au navigateur à l'aide de l'objet de fenêtre ou de document, comme le montre la fonction JavaScript `getSWF()` de cet exemple :

```
<script language="JavaScript">
...
function getSWF(movieName)
{
    if (navigator.appName.indexOf("Microsoft") != -1)
    {
        return window[movieName];
    }
    else
    {
        return document[movieName];
    }
}
...
</script>
```

Si votre script ne détecte pas le type de navigateur de l'utilisateur, ce dernier peut noter un comportement inattendu lors de la lecture des fichiers SWF dans un conteneur HTML.

Exemple : utilisation de l'API externe avec un conteneur ActiveX

Cet exemple illustre l'utilisation de l'API externe pour communiquer entre ActionScript et une application de bureau utilisant le contrôle ActiveX. Il réutilise l'application Introvert IM, y compris le code ActionScript et le fichier SWF. Pour cette raison, aucune description de l'utilisation de l'API externe dans ActionScript n'est fournie. Il est recommandé de se familiariser avec l'exemple précédent pour mieux comprendre celui-ci.

L'application de bureau de cet exemple est écrite en C# à l'aide de Microsoft Visual Studio .NET. L'étude se concentre sur les techniques propres à l'utilisation de l'API externe conjointement avec le contrôle ActiveX. Cet exemple présente les points suivants :

- Appel des fonctions ActionScript à partir d'une application de bureau hébergeant le contrôle ActiveX Flash Player
- Réception des appels de fonction à partir d'ActionScript et traitement de ces appels dans un conteneur ActiveX
- Utilisation d'une classe proxy pour masquer les détails du format XML sérialisé que Flash Player utilise pour les messages envoyés au conteneur ActiveX

Pour obtenir les fichiers d'application de cet exemple, visitez l'adresse www.adobe.com/go/learn_programmingAS3samples_flash_fr. Les fichiers de l'application Introvert IM C# se trouvent dans le dossier Samples/IntrovertIM_CSharp. L'application se compose des fichiers suivants :

Fichier	Description
AppForm.cs	Fichier d'application principal contenant l'interface de Windows Forms C#.
bin/Debug/IntrovertIMApp.swf	Fichier SWF chargé par cette application.
ExternalInterfaceProxy/ExternalInterfaceProxy.cs	Classe servant d'enveloppe au contrôle ActiveX pour la communication avec l'interface externe. Elle fournit les mécanismes d'appel et de réception des appels issus d'ActionScript.
ExternalInterfaceProxy/ExternalInterfaceSerializer.cs	Classe chargée de la conversion des messages au format XML de Flash Player en objets .NET.
ExternalInterfaceProxy/ExternalInterfaceEventArgs.cs	Fichier définissant deux types (classes) C# : une classe delegate personnalisée et une classe event arguments, utilisées par la classe ExternalInterfaceProxy pour notifier à un écouteur un appel de fonction provenant d'ActionScript.
ExternalInterfaceProxy/ExternalInterfaceCall.cs	Cette classe est un objet valeur représentant un appel de fonction d'ActionScript vers le conteneur ActiveX, avec des propriétés destinées au nom de fonction et aux paramètres.
bin/Debug/IntrovertIMApp.swf	Fichier SWF chargé par cette application.
obj/AxInterop.ShockwaveFlashObjects.dll, obj/Interop.ShockwaveFlashObjects.dll	Ensembles d'enveloppes créés par Visual Studio .NET et requis pour accéder au contrôle ActiveX Flash Player (Adobe Shockwave® Flash) à partir du code géré.

Présentation de l'application Introvert IM C#

Cette application exemple est constituée de deux programmes client de messagerie instantanée qui communiquent entre eux ; l'un réside dans un fichier SWF et l'autre est élaboré dans Windows Forms. L'interface utilisateur comprend une occurrence du contrôle ActiveX Shockwave Flash, dans laquelle le fichier SWF contenant le client de messagerie ActionScript est chargé. L'interface comprend également plusieurs champs de texte qui constituent le client de messagerie Windows Forms IM : un champ permet d'entrer des messages (`MessageText`), un autre affiche la transcription des messages échangés par les clients (`Transcript`) et un troisième (`Status`) affiche le statut de disponibilité défini dans le client de messagerie SWF.

Intégration du contrôle ActiveX Shockwave Flash

Pour intégrer le contrôle ActiveX Shockwave Flash dans votre application Windows Forms, vous devez d'abord l'ajouter aux outils de Microsoft Visual Studio.

Pour ajouter le contrôle à la boîte à outils :

1 Ouvrez la boîte à outils Visual Studio.

2 Cliquez du bouton droit sur la section Windows Forms de Visual Studio 2003 ou toute section de Visual Studio 2005. Dans le menu contextuel, sélectionnez Ajouter/Supprimer des éléments dans Visual Studio 2003 (Choisir des éléments dans Visual Studio 2005).

   La boîte de dialogue Personnaliser la boîte à outils (2003) / Choisir les éléments de la boîte à outils (2005) s'ouvre.

3 Sélectionnez l'onglet Composants COM, qui répertorie tous les composants COM disponibles sur votre ordinateur, y compris le contrôle ActiveX Flash Player.

4 Localisez l'option Objet Shockwave Flash et sélectionnez-la.

   Si cette option n'apparaît pas, vérifiez que le contrôle ActiveX Flash Player est installé sur votre système.

Présentation de la communication d'ActionScript vers le conteneur ActiveX

La communication entre l'API externe et une application conteneur ActiveX est semblable à celle établie avec un navigateur Web, avec cependant une importante différence. Comme décrit plus haut, lorsque ActionScript communique avec un navigateur Web, du point de vue du développeur les fonctions sont appelées directement. Les détails du formatage des appels de fonction et des réponses en vue de leur transmission entre le lecteur et le navigateur sont masqués. Toutefois, lorsque l'API externe sert à communiquer avec une application conteneur ActiveX, Flash Player envoie des messages (appels de fonction et valeurs de retour) à l'application dans un format XML spécifique. Elle attend ensuite des appels de fonction et des valeurs de retour d'un format identique de la part de l'application conteneur. Le développeur de l'application conteneur ActiveX doit impérativement écrire un code capable de comprendre et de créer des appels de fonction et des réponses dans le format approprié.

L'exemple Introvert IM C# inclut un jeu de classes qui permet d'éviter le formatage des messages. Vous pouvez alors utiliser les types de données standard pour l'appel de fonctions ActionScript et la réception d'appels de fonction à partir d'ActionScript. Cette fonctionnalité est fournie par la classe ExternalInterfaceProxy, ainsi que d'autres classes d'aide. Ces classes peuvent s'utiliser dans tout projet .NET afin de faciliter la communication avec l'API externe.

Les sections de code suivant, extraites du formulaire principal de l'application (AppForm.cs), illustrent de manière simplifiée l'interaction obtenue grâce à la classe ExternalInterfaceProxy :

```
public class AppForm : System.Windows.Forms.Form
{
    ...
    private ExternalInterfaceProxy proxy;
    ...
    public AppForm()
    {
        ...
        // Register this app to receive notification when the proxy receives
        // a call from ActionScript.
        proxy = new ExternalInterfaceProxy(IntrovertIMApp);
        proxy.ExternalInterfaceCall += new
ExternalInterfaceCallEventHandler(proxy_ExternalInterfaceCall);
        ...
    }
    ...
```

L'application déclare et crée une occurrence de ExternalInterfaceProxy appelée `proxy` en transmettant une référence au contrôle ActiveX Shockwave Flash qui se trouve dans l'interface utilisateur (`IntrovertIMApp`). Ensuite, le code enregistre la méthode `proxy_ExternalInterfaceCall()` afin qu'elle reçoive l'événement `ExternalInterfaceCall` du proxy. Cet événement est distribué par la classe ExternalInterfaceProxy lorsqu'un appel de fonction provient de Flash Player. La souscription à cet événement permet au code C# de recevoir et répondre aux appels de fonction issus d'ActionScript.

Lorsqu'un appel de fonction provient d'ActionScript, l'occurrence de ExternalInterfaceProxy (`proxy`) reçoit l'appel, le convertit au format XML et notifie les objets désignés comme écouteurs de l'événement `ExternalInterfaceCall` du proxy. Dans le cas de la classe AppForm, la méthode `proxy_ExternalInterfaceCall()` gère cet événement, comme suit :

```
/// <summary>
/// Called by the proxy when an ActionScript ExternalInterface call
/// is made by the SWF
/// </summary>
private object proxy_ExternalInterfaceCall(object sender, ExternalInterfaceCallEventArgs e)
{
    switch (e.FunctionCall.FunctionName)
    {
        case "isReady":
            return isReady();
        case "setSWFIsReady":
            setSWFIsReady();
            return null;
        case "newMessage":
            newMessage((string)e.FunctionCall.Arguments[0]);
            return null;
        case "statusChange":
            statusChange();
            return null;
        default:
            return null;
    }
}
...
```

Une occurrence de ExternalInterfaceCallEventArgs (appelée `e` dans cet exemple) est transmise à la méthode. Cet objet possède alors une propriété `FunctionCall` qui est une occurrence de la classe ExternalInterfaceCall.

Une occurrence de ExternalInterfaceCall est un simple objet valeur avec deux propriétés. La propriété `FunctionName` contient le nom de fonction spécifié dans l'instruction `ExternalInterface.Call()` ActionScript. Si des paramètres sont ajoutés dans ActionScript, ils sont inclus dans la propriété `Arguments` de l'objet ExternalInterfaceCall. Dans ce cas, la méthode qui gère l'événement est une simple instruction `switch` qui agit comme gestionnaire de trafic. La valeur de la propriété `FunctionName` (`e.FunctionCall.FunctionName`) détermine quelle méthode de la classe AppForm est appelée.

Dans l'exemple de code précédent, les branches de l'instruction `switch` illustrent des scénarios courants d'appel de fonction. Par exemple, toute méthode doit renvoyer une valeur à ActionScript (ex. l'appel de méthode `isReady()`) ou renvoyer la valeur `null` (comme illustré dans les autres appels de méthode). L'accès aux paramètres transmis à partir d'ActionScript est illustré par l'appel de la méthode `newMessage()` (qui transmet un paramètre `e.FunctionCall.Arguments[0]`, le premier élément du tableau `Arguments`).

L'appel d'une fonction ActionScript à partir de C# à l'aide de la classe ExternalInterfaceProxy se fait de manière encore plus directe que la réception d'un appel de fonction à partir d'ActionScript. Pour appeler une fonction ActionScript, vous utilisez la méthode `Call()` de la classe ExternalInterfaceProxy, comme suit :

```
    /// <summary>
    /// Called when the "Send" button is pressed; the value in the
    /// MessageText text field is passed in as a parameter.
    /// </summary>
    /// <param name="message">The message to send.</param>
    private void sendMessage(string message)
    {
        if (swfReady)
        {
            ...
            // Call the newMessage function in ActionScript.
            proxy.Call("newMessage", message);
        }
    }
    ...
    /// <summary>
    /// Call the ActionScript function to get the current "availability"
    /// status and write it into the text field.
    /// </summary>
    private void updateStatus()
    {
        Status.Text = (string)proxy.Call("getStatus");
    }
    ...
}
```

Comme le montre cet exemple, la méthode `Call()` de la classe ExternalInterfaceProxy est très semblable à son équivalent dans ActionScript, `ExternalInterface.Call()`. Le premier paramètre est une chaîne, le nom de la fonction à appeler. Tout autre paramètre supplémentaire (non illustré ici) est transmis dans la fonction ActionScript. Si cette fonction renvoie une valeur, cette dernière est renvoyée par la méthode `Call()` (comme le montre l'exemple ci-dessus).

Fonctionnement interne de la classe ExternalInterfaceProxy

L'utilisation de l'enveloppe proxy autour du contrôle ActiveX n'est pas toujours pratique ou vous préférerez parfois écrire votre propre classe proxy (par exemple dans un langage de programmation différent ou à destination d'une autre plate-forme). Bien que cette section n'aborde pas tous les détails de la création d'une telle classe, il est intéressant de comprendre le fonctionnement interne de la classe proxy présentée dans cet exemple.

La méthode `CallFunction()` du contrôle ActiveX Shockwave Flash vous permet d'appeler une fonction ActionScript à partir du conteneur ActiveX via l'API externe. Cette procédure est illustrée dans cet extrait tiré de la méthode `Call()` de la classe ExternalInterfaceProxy :

```
// Call an ActionScript function on the SWF in "_flashControl",
// which is a Shockwave Flash ActiveX control.
string response = _flashControl.CallFunction(request);
```

Dans cet extrait de code, `_flashControl` est le contrôle ActiveX Shockwave Flash. Les appels de fonction ActionScript s'effectuent au moyen de la méthode `CallFunction()`. Cette méthode prend un seul paramètre (`request` dans cet exemple) qui correspond à une chaîne contenant les instructions au format XML, c'est-à-dire le nom de la fonction ActionScript à appeler et les paramètres éventuels. Toute valeur renvoyée par ActionScript est convertie en une chaîne au format XML puis renvoyée comme valeur de retour de l'appel `CallFunction()`. Dans cet exemple, la chaîne XML est stockée dans la variable `response`.

La réception d'un appel de fonction issu d'ActionScript est un processus en plusieurs étapes. Les appels de fonction issus d'ActionScript provoquent la distribution de l'événement FlashCall du contrôle ActiveX Shockwave Flash. Ainsi, une classe (telle que ExternalInterfaceProxy) destinée à recevoir des appels d'un fichier SWF doit définir un gestionnaire pour cet événement. Dans la classe ExternalInterfaceProxy, la fonction gestionnaire d'événement est appelée `_flashControl_FlashCall()` et enregistrée dans le constructeur de classe pour écouter l'événement, comme suit :

```
private AxShockwaveFlash _flashControl;

public ExternalInterfaceProxy(AxShockwaveFlash flashControl)
{
    _flashControl = flashControl;
    _flashControl.FlashCall += new
_IShockwaveFlashEvents_FlashCallEventHandler(_flashControl_FlashCall);
}
...
private void _flashControl_FlashCall(object sender, _IShockwaveFlashEvents_FlashCallEvent e)
{
    // Use the event object's request property ("e.request")
    // to execute some action.
    ...
    // Return a value to ActionScript;
    // the returned value must first be encoded as an XML-formatted string.
    _flashControl.SetReturnValue(encodedResponse);
}
```

L'objet événement (`e`) a une propriété `request` (`e.request`) qui correspond à une chaîne contenant des informations sur l'appel de fonction, telles que le nom de fonction et les paramètres, au format XML. Ces informations sont utilisées par le conteneur pour déterminer le code à exécuter. Dans la classe ExternalInterfaceProxy, la requête est convertie du format XML en un objet ExternalInterfaceCall, qui fournit les mêmes éléments, mais sous une forme plus accessible. La méthode `SetReturnValue()` du contrôle ActiveX est utilisée pour renvoyer un résultat de fonction au code ActionScript appelant ; là encore, le paramètre de résultat doit être converti au format XML utilisé par l'API externe.

La communication entre ActionScript et une application hébergeant le contrôle Active X Shockwave Flash nécessite un format XML spécifique qui servira à convertir les appels de fonction et les valeurs. Dans l'exemple Introvert IM C# example, la classe ExternalInterfaceProxy permet au code dans le formulaire de l'application d'agir directement sur des valeurs envoyées ou reçues d'ActionScript, et d'ignorer les détails du format XML utilisé par Flash Player. Pour cela, la classe ExternalInterfaceProxy utilise les méthodes de la classe ExternalInterfaceSerializer pour effectuer la conversion des messages XML en objets .NET. La classe ExternalInterfaceSerializer possède quatre méthodes publiques :

- `EncodeInvoke()` : convertit un nom de fonction et une liste d'instruction C# ArrayList au format XML approprié.
- `EncodeResult()` : convertit une valeur de résultat au format XML approprié.
- `DecodeInvoke()` : décode un appel de fonction issu d'ActionScript. La propriété `request` de l'objet événement FlashCall est transmise à la méthode `DecodeInvoke()` et convertit l'appel en objet ExternalInterfaceCall.
- `DecodeResult()` : décode la chaîne XML reçue comme résultat de l'appel d'une fonction ActionScript.

Ces méthodes convertissent des valeurs C# au format XML de l'API externe et décodent le XML en objets C#. Pour plus de détails sur le format XML utilisé par Flash Player, consultez la section « Format XML de l'API externe » à la page 701.

Chapitre 32 : Sécurité dans Flash Player

La sécurité est une préoccupation essentielle pour Adobe, les utilisateurs, les propriétaires de sites Web et les développeurs de contenu. Pour cette raison, Adobe® Flash® Player inclut un ensemble de règles de sécurité et de contrôles qui protègent l'utilisateur, le propriétaire du site Web et le développeur du contenu. Ce chapitre explique comment travailler avec le modèle de sécurité de Flash Player lors du développement d'applications. Sauf indication contraire, tous les fichiers SWF étudiés dans ce chapitre sont censés être publiés avec ActionScript 3.0 (pour une exécution dans Flash Player 9.0.124.0 ou version ultérieure).

Ce chapitre vise à offrir une présentation générale de la sécurité et non à apporter une explication exhaustive de tous les détails de l'implémentation, des scénarios d'exploitation ou des ramifications de l'utilisation de certaines API. Pour plus de détails sur les concepts de sécurité de Flash Player, consultez la rubrique « Sécurité » du Centre des développeurs de Flash Player à l'adresse www.adobe.com/go/devnet_security_fr.

Pour plus d'informations sur les problèmes de sécurité dans Adobe® AIR™, consultez le chapitre « Sécurité AIR » à l'adresse www.adobe.com/go/learn_air_flash_fr.

Présentation de la sécurité dans Flash Player

L'essentiel de la sécurité de Flash Player repose sur le domaine d'origine des fichiers SWF, médias et autres actifs chargés. Un fichier SWF issu d'un domaine Internet particulier, tel www.exemple.com, peut toujours accéder à l'ensemble des données de ce domaine. Ces actifs sont placés dans le même groupe de sécurité, appelé *sandbox de sécurité*. (Pour plus d'informations, consultez la section « Les sandbox de sécurité » à la page 716.)

Par exemple, un fichier SWF peut charger des fichiers SWF, bitmap, audio, texte et tout autre actif appartenant au même domaine. En outre, la programmation croisée entre deux fichiers SWF d'un même domaine est toujours autorisée, sous réserve qu'ils soient tous deux écrits en ActionScript 3.0. La *programmation croisée* est la capacité d'un fichier SWF à utiliser ActionScript pour accéder aux propriétés, méthodes et objets d'un autre fichier SWF.

Elle n'est pas prise en charge si les fichiers sont écrits en ActionScript 3.0 pour certains et dans des versions antérieures d'ActionScript pour d'autres. La communication entre ces fichiers sera toutefois possible grâce à la classe LocalConnection. Par ailleurs, un fichier SWF ne peut pas, par défaut, accéder par programmation croisée à des fichiers SWF écrits en ActionScript 3.0 appartenant à d'autres domaines ni charger des données à partir d'autres domaines. Ce type d'accès est néanmoins autorisé par le biais d'un appel à la méthode `Security.allowDomain()` dans le fichier SWF chargé. Pour plus de détails sur la programmation croisée, consultez la section « Programmation croisée » à la page 731.

Les règles de sécurité élémentaires présentées ci-après s'appliquent toujours par défaut :

- Des ressources issues du même sandbox de sécurité ont accès les unes aux autres.
- Les fichiers SWF situés dans un sandbox distant ne peuvent jamais accéder aux données et fichiers locaux.

Flash Player considère les domaines suivants comme distincts et définit un sandbox de sécurité pour chacun :

- `http://example.com`
- `http://www.example.com`
- `http://store.example.com`
- `https://www.example.com`

- `http://192.0.34.166`

Même si un domaine nommé, tel que http://example.com, est associé à une adresse IP particulière, par exemple http://192.0.34.166, Flash Player définit un sandbox distinct pour chacun d'eux.

Le développeur dispose de deux méthodes de base pour permettre à un fichier SWF d'accéder à des actifs issus de sandbox différents du sien.

- La méthode `Security.allowDomain()` (consultez la section « Contrôles de création (développeur) » à la page 724)
- Le fichier de régulation d'URL (consultez la section « Contrôles de site Web (fichiers de régulation) » à la page 721)

Le modèle de sécurité Flash Player établit une distinction entre le chargement de contenu et l'accès à des données ou leur extraction. Le *contenu* correspond à des médias, notamment les éléments visuels que Flash Player peut afficher, les fichiers audio et vidéo et les fichiers SWF contenant des médias affichés. Les *données* sont des éléments uniquement accessibles au code ActionScript. Le contenu et les données ne sont pas chargés de la même façon.

- Chargement de contenu : vous pouvez charger du contenu à l'aide de classes telles que Loader, Sound et NetStream.
- Extraction de données : il est possible d'extraire des données à partir de médias chargés grâce aux objets Bitmap, à la méthode `BitmapData.draw()`, à la propriété `Sound.id3` ou à la méthode `SoundMixer.computeSpectrum()`.
- Accès à des données : vous pouvez accéder à des données directement en les chargeant à partir d'un fichier externe (un fichier XML, par exemple) par le biais de classes telles que URLStream, URLLoader, Socket et XMLSocket.

Le modèle de sécurité Flash Player définit différentes règles concernant le chargement de contenu et l'accès aux données. En général, les restrictions sont moindres sur le chargement de contenu que sur l'accès aux données.

Habituellement, le contenu (fichiers SWF, bitmap, mp3 et vidéo) peut être chargé de n'importe quelle source mais s'il provient d'un domaine différent du fichier SWF à l'origine du chargement, ce contenu se trouvera dans un sandbox de sécurité distinct.

Certains obstacles s'appliquent au chargement de contenu :

- Par défaut, les fichiers SWF locaux (ceux chargés à partir d'une adresse ne se trouvant pas sur un réseau, par exemple le disque dur de l'utilisateur) sont classés dans le sandbox local avec système de fichiers. Ces fichiers ne peuvent pas charger de contenu provenant d'un réseau. Pour plus d'informations, consultez la section « Les sandbox locaux » à la page 716.
- Les serveurs RTMP (Real-Time Messaging Protocol) peuvent limiter l'accès au contenu. Pour plus d'informations, consultez la section « Contenu diffusé à l'aide de serveurs RTMP » à la page 731.

Si le média chargé est une image, un fichier audio ou une vidéo, ses données (par exemple données de pixels ou sons) sont accessibles pour un fichier SWF situé en dehors de son sandbox de sécurité, à condition que le domaine de ce fichier SWF ait été inclus dans un fichier de régulation d'URL dans le domaine d'origine du média. Pour plus d'informations, consultez la section « Accès aux médias chargés comme s'il s'agissait de données » à la page 734.

Les données chargées peuvent également provenir de fichiers texte ou XML chargés avec l'objet URLLoader. Là encore, l'accès à des données situées dans un autre sandbox de sécurité nécessite l'attribution des autorisations suffisantes par le biais d'un fichier de régulation d'URL placé dans le domaine d'origine. Pour plus d'informations, consultez la section « Utilisation de URLLoader et URLStream » à la page 736.

Les sandbox de sécurité

Les ordinateurs client peuvent obtenir des fichiers SWF individuels de diverses sources, par exemple d'un site Web externe ou d'un système de fichiers local. Flash Player associe chaque fichier SWF et chaque ressource (objets partagés, fichiers bitmap, son, vidéo et données) à des sandbox de sécurité en fonction de leur origine au moment du chargement dans Flash Player. Les sections suivantes décrivent les règles mises en place par Flash Player pour contrôler ce à quoi un fichier SWF peut accéder au sein d'un sandbox donné.

Pour plus de détails sur la sécurité de Flash Player, consultez la rubrique « Sécurité » du Centre des développeurs Flash Player à l'adresse www.adobe.com/go/devnet_security_fr.

Les sandbox distants

Flash Player classe les actifs (y compris les fichiers SWF) en provenance d'Internet dans des sandbox différents correspondant à leur domaine d'origine. Par défaut, ces fichiers sont autorisés à accéder à toutes les ressources issues de leur propre serveur. Il est possible d'autoriser les fichiers SWF distants à accéder à des données d'autres domaines à l'aide d'autorisations explicites portant sur les sites Web et les auteurs, par exemple des fichiers de régulation d'URL et la méthode `Security.allowDomain()`. Pour plus d'informations, consultez les sections « Contrôles de site Web (fichiers de régulation) » à la page 721 et « Contrôles de création (développeur) » à la page 724.

Les fichiers SWF distants ne peuvent pas charger de fichiers ou de ressources locales.

Pour plus de détails sur la sécurité de Flash Player, consultez la rubrique « Sécurité » du Centre des développeurs Flash Player à l'adresse www.adobe.com/go/devnet_security_fr.

Les sandbox locaux

Un fichier est dit *local* s'il est référencé par le biais du protocole `file:` ou d'un chemin UNC (Universal Naming Convention). Les fichiers SWF locaux sont placés dans l'un de quatre sandbox locaux :

- Sandbox local avec système de fichiers : pour des raisons de sécurité, Flash Player place par défaut tous les fichiers SWF et actifs locaux dans ce sandbox. De là, les fichiers SWF peuvent lire les fichiers locaux (à l'aide de la classe URLLoader, par exemple), mais en aucun cas communiquer avec le réseau. Ceci garantit à l'utilisateur que les données locales ne peuvent pas filtrer hors du réseau ou autrement être partagées de manière inopportune.
- Sandbox local avec réseau : lors de la compilation d'un fichier SWF, vous pouvez spécifier s'il dispose d'un accès réseau lorsqu'il est exécuté comme fichier local (voir « Définition du type de sandbox pour les fichiers SWF locaux » à la page 717). De tels fichiers sont placés dans le sandbox local avec réseau. Les fichiers SWF placés dans le sandbox local avec réseau abandonnent leur accès aux fichiers locaux. En échange, ils sont autorisés à accéder aux données sur le réseau. Toutefois, un fichier local avec réseau ne peut pas lire des données dérivées du réseau si aucune autorisation n'est accordée pour cela par le biais d'un fichier de régulation d'URL ou d'un appel à la méthode`Security.allowDomain()`. A cet effet, le fichier de régulation d'URL doit accorder une autorisation à *tous* les domaines en utilisant `allow-access-from domain="*"/` ou `Security.allowDomain("*")`. Pour plus d'informations, consultez les sections « Contrôles de site Web (fichiers de régulation) » à la page 721 et « Contrôles de création (développeur) » à la page 724.
- Sandbox local approuvé : les fichiers SWF locaux enregistrés comme approuvés (par l'utilisateur ou un programme d'installation) sont placés dans ce sandbox. Les administrateurs système et les utilisateurs peuvent aussi associer un fichier SWF local au sandbox local approuvé ou l'en dissocier, selon les contraintes de sécurité (voir « Contrôles administrateur » à la page 718 et « Contrôles utilisateur » à la page 720). Les fichiers SWF associés au Sandbox local approuvé peuvent interagir avec tous les autres fichiers SWF et charger des données à partir de n'importe quel emplacement (à distance ou localement).

- Sandbox de sécurité de l'application AIR : ce sandbox contient du contenu installé à l'aide de l'application AIR en cours d'exécution. Par défaut, les fichiers du sandbox de sécurité de l'application AIR peuvent accéder par programmation croisée aux fichiers de n'importe quel domaine. En revanche, les fichiers se trouvant en dehors de ce sandbox ne peuvent pas accéder par programmation croisée au fichier AIR. Par défaut, les fichiers du sandbox de sécurité de l'application AIR peuvent charger le contenu et les données de n'importe quel domaine.

La communication entre le sandbox local avec réseau et le sandbox local avec système de fichiers est strictement interdite, tout comme la communication entre le sandbox local avec système de fichiers et le sandbox distant. Elles ne peuvent pas être autorisées par une application Flash Player ni par un utilisateur ou un administrateur.

La programmation croisée entre les fichiers HTML et SWF locaux (par exemple à l'aide de la classe ExternalInterface) exige que les deux fichiers impliqués se trouvent dans le sandbox local approuvé. Cette contrainte vient du fait que les modèles de sécurité locaux des navigateurs diffèrent de celui de Flash Player.

Les fichiers SWF du sandbox local avec réseau ne peuvent pas charger des fichiers SWF du sandbox local avec système de fichiers. Les fichiers SWF du sandbox local avec système de fichier ne peuvent pas charger des fichiers SWF du sandbox local avec réseau.

Définition du type de sandbox pour les fichiers SWF locaux

Vous pouvez configurer un fichier SWF pour le sandbox local avec système de fichiers ou le sandbox local avec réseau en définissant les paramètres de publication du document dans l'outil de programmation.

Un utilisateur ou l'administrateur d'un ordinateur peut spécifier si un fichier SWF local est approuvé, lui permettant ainsi de charger des données de tous les domaines, locaux ou réseau. Cette caractéristique est définie dans les répertoires Flash Player Trust global et utilisateur. Pour plus d'informations, consultez les sections « Contrôles administrateur » à la page 718 et « Contrôles utilisateur » à la page 720.

Pour plus d'informations sur les sandbox locaux, consultez la section « Les sandbox locaux » à la page 716.

Propriété Security.sandboxType

La propriété statique en lecture seule `Security.sandboxType` permet à l'auteur d'un fichier SWF de déterminer le type de sandbox auquel Flash Player a associé le fichier SWF. La classe Security inclut des constantes qui représentent les valeurs possibles de la propriété `Security.sandboxType`, comme suit :

- `Security.REMOTE` : le fichier SWF provient d'une URL Internet et son fonctionnement est régi par les règles de sandbox de domaines.
- `Security.LOCAL_WITH_FILE` : le fichier SWF est un fichier local mais n'a pas été approuvé par l'utilisateur ni publié avec une désignation réseau. Le fichier SWF peut lire les sources de données locales mais ne peut pas communiquer avec Internet.
- `Security.LOCAL_WITH_NETWORK` : le fichier SWF est un fichier local non approuvé par l'utilisateur mais qui a été publié avec une désignation réseau. Le fichier SWF peut communiquer sur Internet mais ne peut pas lire les sources de données locales.
- `Security.LOCAL_TRUSTED` : le fichier SWF est un fichier local qui a été approuvé par l'utilisateur via le Gestionnaire des paramètres ou le fichier de configuration Flash Player Trust. Ce fichier SWF peut lire les sources de données locales et communiquer avec Internet.
- `Security.APPLICATION` : le fichier SWF est exécuté dans une application AIR, et a été installé avec le package (le fichier AIR) pour cette application. Par défaut, les fichiers du sandbox de sécurité de l'application AIR peuvent accéder par programmation croisée aux fichiers de n'importe quel domaine. En revanche, les fichiers se trouvant en dehors de ce sandbox ne peuvent pas accéder par programmation croisée au fichier AIR. Par défaut, les fichiers du sandbox de sécurité de l'application AIR peuvent charger le contenu et les données de n'importe quel domaine.

Contrôles des autorisations

Le modèle de sécurité d'exécution du client Flash Player repose sur les ressources, c'est-à-dire des objets tels que des fichiers SWF, des données locales et des URL Internet. Les *parties prenantes* détiennent ou utilisent ces ressources. Elles peuvent exercer un contrôle (via des paramètres de sécurité) sur leurs propres ressources, chaque ressource ayant quatre parties prenantes. Flash Player applique une stricte hiérarchie d'autorité sur ces contrôles, comme le montre l'illustration suivante :

Hiérarchie des contrôles de sécurité

Ainsi, par exemple, si un administrateur restreint l'accès à une ressource, aucune autre partie prenante ne peut revenir sur cette restriction.

Contrôles administrateur

L'administrateur d'un ordinateur (un utilisateur ayant ouvert une session avec des droits d'administration) peut définir des paramètres de sécurité Flash Player qui s'appliquent à tous les utilisateurs de l'ordinateur. Dans un environnement autre que l'entreprise, par exemple un ordinateur domestique, un utilisateur dispose aussi d'un accès administrateur. Même au sein d'une entreprise, certains utilisateurs peuvent posséder des droits d'administration sur l'ordinateur.

Il existe deux types de contrôles administrateur :

- Fichier mms.cfg
- Répertoire Flash Player Trust global

Fichier mms.cfg

Le fichier mms.cfg est un fichier texte qui permet à l'administrateur d'autoriser ou de limiter l'accès à diverses fonctionnalités. Lors du lancement de Flash Player, l'application lit les paramètres de sécurité dans ce fichier, puis les utilise pour limiter la fonctionnalité. Le fichier mms.cfg contient des paramètres utilisés par l'administrateur pour gérer des fonctionnalités telles que les contrôles de confidentialité, la sécurité des fichiers locaux, les connexions socket, etc.

Un fichier SWF peut obtenir des informations sur les fonctions désactivées en appelant les propriétés `Capabilities.avHardwareDisable` et `Capabilities.localFileReadDisable`. Toutefois, la plupart des paramètres du fichier mms.cfg ne peuvent être interrogés à partir d'ActionScript.

Pour mettre en place sur un ordinateur des stratégies de confidentialité et de sécurité indépendantes des applications, le fichier mms.cfg doit être uniquement modifié par un administrateur système. Le fichier mms.cfg n'est pas destiné aux programmes d'installation des applications. Bien qu'un programme d'installation exécuté avec des privilèges administrateur puisse modifier le contenu du fichier mms.cfg, Adobe considère cette pratique comme une violation de la confiance de l'utilisateur et presse les créateurs de programmes d'installation de ne jamais modifier le fichier mms.cfg.

Le fichier mms.cfg est enregistré dans le chemin suivant :

- Windows : *system*\Macromed\Flash\mms.cfg

  (par exemple, C:\windows\system32\Macromed\Flash\mms.cfg)

- Mac : *app support*/Macromedia/mms.cfg

  (par exemple, /Bibliothèque/Application Support/Macromedia/mms.cfg)

Pour plus d'informations sur le fichier mms.cfg, consultez le Guide d'administration de Flash Player à l'adresse suivante : www.adobe.com/go/flash_player_admin_fr.

Répertoire Flash Player Trust global

Les administrateurs et les programmes d'installation peuvent approuver des fichiers SWF locaux spécifiques pour tous les utilisateurs. Ces fichiers SWF sont associés au sandbox local approuvé. Ils peuvent interagir avec tout autre fichier SWF puisqu'ils peuvent charger des données stockées localement ou à distance. Les fichiers approuvés dans le répertoire Flash Player Trust global se trouvent dans le chemin suivant :

- Windows : *system*\Macromed\Flash\FlashPlayerTrust

  (par exemple, C:\WINDOWS\system32\Macromed\Flash\FlashPlayerTrust)

- Mac : *app support*/Macromedia/FlashPlayerTrust

  (par exemple, /Bibliothèque/Application Support/Macromedia/FlashPlayerTrust)

Le répertoire Flash Player Trust peut contenir un nombre illimité de fichiers texte, chacun répertoriant des chemins d'accès approuvés, à raison d'un chemin par ligne. Chaque chemin d'accès peut mener à un fichier SWF ou HTML, ou à un répertoire. Les lignes de commentaire commencent par le symbole `#`. Par exemple, un fichier de configuration Flash Player Trust contenant le texte suivant approuve tous les fichiers placés dans le répertoire spécifié et ses sous-répertoires :

```
# Trust files in the following directories:
C:\Documents and Settings\All Users\Documents\SampleApp
```

Les chemins répertoriés dans un fichier de configuration Trust doivent toujours faire référence au système local ou à un réseau SMB. Les chemins d'accès HTTP placés dans un fichier de configuration Trust sont ignorés; seuls les fichiers locaux peuvent être approuvés.

Pour éviter les conflits, attribuez au fichier de configuration Trust un nom correspondant à l'application installée, suivi de l'extension de fichier .cfg.

En tant que développeur diffusant un fichier SWF à exécuter localement par le biais d'un programme d'installation, vous pouvez faire en sorte que ce programme d'installation ajoute un fichier de configuration au répertoire Flash Player Trust global afin d'accorder des privilèges complets au fichier que vous diffusez. Le programme d'installation doit être exécuté par un utilisateur doté de droits d'administration. Contrairement au fichier mms.cfg, le répertoire Flash Player Trust global est prévu pour les programmes d'installation devant accorder des autorisations. Il permet aux administrateurs et aux programmes d'installation de désigner des applications locales approuvées.

Il existe également des répertoires Flash Player Trust destinés aux utilisateurs individuels (voir « Contrôles utilisateur » à la page 720).

Contrôles utilisateur

Flash Player propose trois mécanismes de définition des autorisations au niveau utilisateur : l'interface de paramétrage, le Gestionnaire des paramètres et le répertoire Flash Player Trust utilisateur.

Interface de paramétrage et Gestionnaire des paramètres

L'interface de paramétrage est un mécanisme interactif qui permet de configurer rapidement les paramètres d'un domaine donné. Le Gestionnaire des paramètres, avec son interface plus détaillée, permet d'effectuer des modifications globales sur les autorisations de plusieurs domaines ou de tous. En outre, si un fichier SWF nécessite une nouvelle autorisation qui oblige à prendre des décisions en cours d'exécution concernant la sécurité ou la confidentialité, des boîtes de dialogue s'affichent dans lesquelles les utilisateurs peuvent régler certains paramètres de Flash Player.

Le Gestionnaire des paramètres et l'interface de paramétrage proposent des options associées à la sécurité telles que les paramètres de la caméra et du microphone, les paramètres de stockage d'objets partagés, les paramètres relatifs au contenu hérité, etc.

Remarque : *les réglages effectués dans le fichier mms.cfg (voir « Contrôles administrateur » à la page 718) ne se reflètent pas dans le Gestionnaire des paramètres.*

Pour plus d'informations sur le Gestionnaire des paramètres, visitez l'adresse www.adobe.com/go/settingsmanager_fr.

Répertoire Flash Player Trust utilisateur

Les utilisateurs et programmes d'installation peuvent approuver des fichiers SWF locaux spécifiques. Ces fichiers SWF sont associés au sandbox local approuvé. Ils peuvent interagir avec tout autre fichier SWF puisqu'ils peuvent charger des données stockées localement ou à distance. Pour approuver un fichier, l'utilisateur le désigne dans le répertoire Player Trust utilisateur, qui se trouve dans le même répertoire que la zone de stockage des objets partagés Flash, aux emplacements suivants (ces emplacements sont spécifiques à l'utilisateur actif) :

- Windows : app data\Macromedia\Flash Player\#Security\FlashPlayerTrust

  (par exemple, C:\Documents and Settings\JohnD\Application Data\Macromedia\Flash Player\#Security\FlashPlayerTrust sous Windows XP ou C:\Users\JohnD\AppData\Roaming\Macromedia\Flash Player\#Security\FlashPlayerTrust sous Windows Vista)

  Sous Windows, le dossier Application Data est caché par défaut. Pour afficher les dossiers et fichiers cachés, sélectionnez le Poste de travail pour ouvrir l'Explorateur Windows, choisissez Outils > Options des dossiers et sélectionnez l'onglet Affichage. Sous l'onglet Affichage, sélectionnez le bouton d'option Afficher les fichiers et les dossiers cachés.

- Mac : app data/Macromedia/Flash Player/#Security/FlashPlayerTrust

  (par exemple, /Utilisateurs/JohnD/Bibliothèque/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust)

  Ces paramètres s'appliquent uniquement à l'utilisateur actif, et non aux autres utilisateurs qui ouvrent une session sur l'ordinateur. Si un utilisateur sans droits d'administration installe une application dans sa partie réservée du système, le répertoire Flash Player Trust utilisateur permet au programme d'installation d'enregistrer cette application comme approuvée pour l'utilisateur en question.

En tant que développeur diffusant un fichier SWF à exécuter localement par le biais d'un programme d'installation, vous pouvez faire en sorte que ce programme d'installation ajoute un fichier de configuration au répertoire Flash Player Trust utilisateur afin d'accorder des privilèges complets au fichier que vous diffusez. Même dans ce cas de figure, le fichier placé dans le répertoire Flash Player Trust utilisateur constitue un contrôle utilisateur puisque son initialisation résulte d'une action de l'utilisateur (l'installation).

Il existe également un répertoire Flash Player Trust global utilisé par l'administrateur ou les programmes d'installation pour enregistrer une application destinée à l'ensemble des utilisateurs d'un ordinateur (voir « Contrôles administrateur » à la page 718).

Contrôles de site Web (fichiers de régulation)

Si vous souhaitez rendre les données d'un serveur Web accessibles aux fichiers SWF issus de domaines différents, vous pouvez créer un fichier de régulation sur votre serveur. Un *fichier de régulation* est un fichier XML résidant à un emplacement spécifique sur votre serveur.

Les fichiers de régulation ont une incidence sur l'accès à certains actifs, notamment les suivants :

- Les données de fichiers bitmap, son et vidéo
- Le chargement des fichiers XML et texte
- L'importation de fichiers SWF à partir d'autres domaines de sécurité dans le domaine du fichier SWF à l'origine du chargement
- L'accès aux connexions socket et socket XML

Les objets ActionScript instancient deux types de connexions serveur : les connexions serveur liées à des documents et les connexions socket. Les objets ActionScript tels que Loader, Sound, URLLoader et URLStream instancient des connexions serveur liées à des documents et chargent un fichier à partir d'une URL. Les objets ActionScript Socket et XMLSocket établissent des connexions socket, qui fonctionnent avec des flux de données et non des documents chargés.

Flash Player prenant en charge deux types de connexions serveur, il existe deux types de fichiers de régulation : les fichiers de régulation d'URL et les fichiers de régulation socket.

- Les connexions liées à des documents exigent des *fichiers de régulation d'URL*. Ces fichiers permettent au serveur d'indiquer que ses données et documents sont accessibles aux fichiers SWF servis à partir de certains domaines ou de tous les domaines.
- Les connexions socket requièrent des *fichiers de régulation socket*, qui permettent un accès réseau direct au niveau socket TCP le plus bas, à l'aide des classes Socket et XMLSocket.

Flash Player exige que les fichiers de régulation soient transmis par le biais du protocole utilisé par la tentative de connexion. Par exemple, si vous placez un fichier de régulation sur un serveur HTTP, les fichiers SWF issus d'autres domaines sont autorisés à charger les données qu'il contient en tant que serveur HTTP. Cependant, si vous ne fournissez aucun fichier de régulation socket sur ce même serveur, vous empêchez les fichiers SWF issus d'autres domaines d'accéder au serveur au niveau socket. La méthode de récupération du fichier de régulation doit donc correspondre à la méthode de connexion.

L'utilisation et la syntaxe des fichiers de régulation, tels qu'ils s'appliquent aux fichiers SWF publiés pour Flash Player 10, sont décrites dans les paragraphes suivants. (L'implémentation des fichiers de régulation est légèrement différente dans les versions antérieures de Flash Player, dont la sécurité a été renforcée ultérieurement.) Pour plus d'informations sur les fichiers de régulation, consultez le chapitre « Modifications du fichier de régulation dans Flash Player 9 » dans le Centre des développeurs Flash Player à l'adresse www.adobe.com/go/devnet_security_fr.

Fichiers de régulation maître

Par défaut, Flash Player (et le contenu AIR qui ne figure pas dans le sandbox d'application AIR) commence par rechercher le fichier de régulation d'URL `crossdomain.xml` dans le répertoire racine du serveur, puis recherche un fichier de régulation socket sur le port 843. Tout fichier résidant à l'un de ces emplacements constitue le *fichier de régulation maître*. (Dans le cas des connexions socket, Flash Player recherche également un fichier de régulation socket sur le port utilisé par la connexion principale. Un fichier de régulation se trouvant sur ce port ne constitue cependant pas un fichier de régulation maître.)

Outre la définition des autorisations d'accès, le fichier de régulation maître peut également contenir une instruction *meta-policy*. Une méta-régulation détermine les emplacements auxquels peuvent résider les fichiers de régulation. La méta-régulation par défaut des fichiers de régulation d'URL correspond à « master-only » ; autrement dit, /crossdomain.xml est le seul fichier de régulation autorisé sur le serveur. La méta-régulation par défaut des fichiers de régulation socket correspond à « all » ; autrement dit, tout socket figurant sur l'hôte peut servir un fichier de régulation de socket.

Remarque : *dans Flash Player 9 et les versions antérieures, la méta-régulation par défaut des fichiers de régulation d'URL correspondait à « all ». Tout répertoire peut donc contenir un fichier de régulation. Si vous avez déployé des applications qui chargent d'autres fichiers de régulation que le fichier /crossdomain.xml par défaut, et que ces applications sont maintenant susceptibles de s'exécuter dans Flash Player 10, vous (ou l'administrateur du serveur) devez modifier le fichier de régulation maître pour autoriser l'utilisation d'autres fichiers de régulation. Pour plus d'informations sur la définition d'une méta-régulation différente, consultez le chapitre « Modifications du fichier de régulation dans Flash Player 9 » dans le Centre des développeurs Flash Player à l'adresse www.adobe.com/go/devnet_security_fr.*

Un fichier SWF peut effectuer une recherche sur un nom de fichier de régulation ou un emplacement différent en appelant la méthode `Security.loadPolicyFile()`. Cependant, si le fichier de régulation maître ne spécifie pas que l'emplacement cible peut servir des fichiers de régulation, l'appel à `loadPolicyFile()` est sans effet, même si un fichier de régulation se trouve à l'emplacement en question. Appelez `loadPolicyFile()` avant d'exécuter toute opération réseau requérant le fichier de régulation. Flash Player place automatiquement les requêtes réseau derrière les tentatives de requête de fichier de régulation correspondantes. Il est donc acceptable, par exemple, d'appeler `Security.loadPolicyFile()` immédiatement avant de lancer une opération réseau.

Lorsqu'il recherche un fichier de régulation maître,. Flash Player attend une réponse du serveur pendant trois secondes. En l'absence d'une réponse, l'application considère comme acquis qu'il n'existe pas de fichier de régulation maître. Toutefois, si aucune valeur de dépassement de délai par défaut est définie pour les appels à `loadPolicyFile()`, Flash Player suppose que le fichier appelé existe et attend aussi longtemps que nécessaire pour le charger. Pour avoir la certitude qu'un fichier de régulation maître est chargé, appelez-le donc explicitement par le biais de `loadPolicyFile()`.

Bien que la méthode s'appelle `Security.loadPolicyFile()`, aucun fichier de régulation n'est chargé tant qu'un appel réseau requérant un tel fichier n'a pas été émis. Les appels à `loadPolicyFile()` indiquent simplement à Flash Player où rechercher les fichiers de régulation, le cas échéant.

Aucune notification indiquant l'initiation ou la fin d'une requête de fichier de régulation n'est envoyée, car cela n'est pas nécessaire. Flash Player recherche les fichiers de régulation de manière asynchrone et attend automatiquement que ces recherches aient abouti pour établir des connexions.

Les sections suivantes s'appliquent uniquement aux fichiers de régulation d'URL. Pour plus d'informations sur les fichiers de régulation socket, consultez la section « Connexion aux sockets » à la page 736.

Portée des fichiers de régulation d'URL

Un fichier de régulation d'URL s'applique uniquement au répertoire dans lequel il est chargé et à ses sous-répertoires. Un fichier de régulation placé dans le répertoire racine s'applique à l'ensemble du serveur. En revanche, un fichier de régulation chargé d'un sous-répertoire quelconque s'applique uniquement à celui-ci et à ses sous-répertoires.

Un fichier de régulation contrôle uniquement l'accès au serveur sur lequel il réside. Par exemple, un fichier de régulation situé dans https://www.adobe.com:8080/crossdomain.xml ne s'applique qu'aux appels de chargement de données passés vers www.adobe.com sur HTTPS au port 8080.

Définition des autorisations d'accès dans un fichier de régulation d'URL

Un fichier de régulation contient une seule balise `cross-domain-policy`, qui contient elle-même aucune ou plusieurs balises `allow-access-from`. Chaque balise `allow-access-from` contient un attribut `domain` qui spécifie une adresse IP exacte, un domaine exact ou un domaine générique (tout domaine). Les domaines génériques sont indiqués de deux façons :

- Par un astérisque (*) seul, qui représente tous les domaines et toutes les adresses IP
- Par un astérisque suivi d'un suffixe, qui représente uniquement les domaines se terminant par ce suffixe

Les suffixes doivent commencer par un point. Cependant, les domaines génériques suivis de suffixes peuvent correspondre à des domaines qui sont composés uniquement du suffixe sans le point de séparation. xyz.com, par exemple, fait partie de *.xyz.com. L'utilisation de caractères génériques est interdite dans les spécifications de domaine IP.

L'exemple suivant présente un fichier de régulation d'URL qui autorise l'accès à des fichiers SWF issus des domaines *.example.com, www.friendOfExample.com et 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>
```

Si vous spécifiez une adresse IP, seuls les fichiers SWF chargés depuis cette adresse IP à l'aide de la syntaxe IP (par exemple, http://65.57.83.12/flashmovie.swf) sont accessibles ; les fichiers chargés à l'aide d'une syntaxe domaine-nom sont inaccessibles Flash Player n'effectue pas de résolution DNS.

Vous pouvez autoriser l'accès à des documents provenant de tout autre domaine, comme indiqué dans l'exemple suivant :

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
<allow-access-from domain="*" />
</cross-domain-policy>
```

Chaque balise `allow-access-from` peut présenter l'attribut facultatif `secure`, dont la valeur par défaut est `true`. Si votre fichier de régulation se trouve sur un serveur HTTPS et que vous voulez autoriser les fichiers SWF résidant sur un autre type de serveur à charger des données à partir du serveur HTTPS, vous pouvez définir l'attribut sur `false`.

La définition de l'attribut `secure` sur `false` risque de compromettre la sécurité fournie par le protocole HTTPS. Plus particulièrement, la définition de cet attribut sur `false` rend le contenu sécurisé vulnérable aux attaques d'espionnage. Adobe recommande vivement de ne pas définir l'attribut `secure` sur `false`.

Si les données à charger se trouvent sur un serveur HTTPS, alors que le fichier SWF à l'origine du chargement réside sur un serveur HTTP, Adobe recommande le transfert du fichier SWF sur un serveur HTTPS. Ce faisant, toutes les copies de vos données sécurisées sont protégées par HTTPS. Cependant si vous décidez que vous devez conserver le fichier SWF à l'origine du chargement sur un serveur HTTP, ajoutez l'attribut `secure="false"` à la balise `allow-access-from`, comme le montre le code suivant :

```
<allow-access-from domain="www.example.com" secure="false" />
```

Pour autoriser l'accès, vous pouvez aussi utiliser la balise `allow-http-request-headers-from`. Elle permet à un client hébergeant du contenu issu d'un autre domaine d'autorisation d'envoyer des en-têtes définis par l'utilisateur à votre domaine. La balise `<allow-access-from>` autorise d'autres domaines à récupérer des données de votre domaine, tandis que la balise `allow-http-request-headers-from` permet à d'autres domaines d'envoyer des données, sous forme d'en-têtes, à votre domaine. Dans l'exemple ci-dessous, n'importe quel domaine est autorisé à envoyer l'en-tête SOAPAction au domaine en cours :

```
<cross-domain-policy>
    <allow-http-request-headers-from domain="*" headers="SOAPAction"/>
</cross-domain-policy>
```

Si l'instruction `allow-http-request-headers-from` figure dans le fichier de régulation maître, elle s'applique à tous les répertoires de l'hôte. Dans le cas contraire, elle s'applique uniquement au répertoire, et sous-répertoires correspondants, du fichier de régulation contenant l'instruction.

Préchargement des fichiers de régulation

Le chargement des données à partir d'un serveur ou la connexion à un socket sont des opérations asynchrones. Flash Player attend simplement la fin du téléchargement du fichier de régulation avant de lancer l'opération principale. En revanche, l'extraction de données de pixel d'images ou de données d'échantillonnage de sons est une opération synchrone. Le fichier de régulation doit donc être chargé pour que vous puissiez extraire les données. Lors du chargement du média, spécifiez que le fichier de régulation doit être recherché :

- Si vous utilisez la méthode `Loader.load()`, définissez la propriété `checkPolicyFile` du paramètre `context`, qui constitue un objet LoaderContext.
- Si vous incorporez une image dans un champ de texte à l'aide de la balise `<img>`, définissez l'attribut `checkPolicyFile` de la balise `<img>` sur `true`, comme dans l'exemple suivant :

  ```
  <img checkPolicyFile = "true" src = "example.jpg">
  ```

- Si vous utilisez la méthode `Sound.load()`, définissez la propriété `checkPolicyFile` du paramètre `context`, qui constitue un objet SoundLoaderContext.
- Si vous utilisez la classe NetStream, définissez la propriété `checkPolicyFile` de l'objet NetStream.

Lorsque vous définissez l'un de ces paramètres, Flash Player commence par vérifier si des fichiers de régulation ont déjà été téléchargés pour ce domaine. Il recherche ensuite le fichier de régulation à l'emplacement par défaut sur le serveur, puis les instructions `<allow-access-from>` et une méta-régulation. En dernier lieu, il contrôle tout appel en attente à la méthode `Security.loadPolicyFile()` pour vérifier s'il s'applique.

Contrôles de création (développeur)

L'API ActionScript principale utilisée dans l'attribution des privilèges de sécurité est la méthode `Security.allowDomain()`, qui accorde des droits aux fichiers SWF du domaine que vous spécifiez. Dans l'exemple suivant, un fichier SWF autorise l'accès des fichiers SWF servis à partir du domaine www.example.com :

```
Security.allowDomain("www.example.com")
```

Cette méthode autorise les opérations suivantes :

- Programmation croisée entre fichiers SWF (voir« Programmation croisée » à la page 731)
- Accès à la liste d'affichage (voir « Parcours de la liste d'affichage » à la page 733)
- Détection d'événements (voir « Sécurité des événements » à la page 734)
- Accès total aux propriétés et méthodes de l'objet Stage (voir « Sécurité de la scène » à la page 732)

La méthode `Security.allowDomain()` sert avant tout à permettre aux fichiers SWF situés dans un domaine externe d'effectuer une programmation croisée avec le fichier SWF qui appelle la méthode `Security.allowDomain()`. Pour plus de détails sur la programmation croisée, consultez la section « Programmation croisée » à la page 731.

La spécification de l'adresse IP en tant que paramètre de `Security.allowDomain()` n'autorise pas l'accès de toutes les parties provenant de l'adresse IP spécifiée. Au contraire, elle autorise l'accès d'une partie présentant une URL identique à l'adresse IP spécifiée, plutôt qu'un nom de domaine renvoyant à l'adresse IP. Par exemple, si le nom de domaine www.example.com renvoie à l'adresse IP 192.0.34.166, un appel à `Security.allowDomain("192.0.34.166")` ne donne pas accès à www.example.com.

Vous pouvez transmettre le caractère générique `"*"` à la méthode `Security.allowDomain()` pour permettre l'accès à partir de tous les domaines. Soyez prudent lorsque vous utilisez le caractère générique "*" car celui-ci autorise les fichiers SWF issus de `tous` les domaines à effectuer une programmation dans le fichier SWF appelant.

ActionScript inclut une seconde API d'autorisation appelée `Security.allowInsecureDomain()`. Cette méthode joue le même rôle que la méthode `Security.allowDomain()` à la différence suivante : lorsqu'elle est appelée à partir d'un fichier SWF servi par une connexion HTTPS, elle autorise l'accès à ce fichier appelant pour d'autres fichiers SWF servis à partir d'un protocole non sécurisé, tel HTTP. Cependant, il est déconseillé de permettre la programmation croisée entre des fichiers issus d'un protocole sécurisé et ceux provenant d'un protocole qui ne l'est pas. Cette pratique est susceptible de rendre le contenu vulnérable aux attaques d'espionnage. Ces attaques se déroulent comme suit, par exemple : puisque la méthode `Security.allowInsecureDomain()` permet aux fichiers SWF servis sur des connexions HTTP d'accéder aux données HTTPS sécurisées, un attaquant qui s'interposerait entre le serveur HTTP et les utilisateurs pourraient remplacer votre fichier SWF HTTP par un fichier de son cru afin d'accéder à vos données HTTPS.

Autre méthode importante liée à la sécurité, `Security.loadPolicyFile()` permet à Flash Player de rechercher le fichier de régulation à un autre emplacement. Pour plus d'informations, consultez la section « Contrôles de site Web (fichiers de régulation) » à la page 721.

Restriction des API de réseau

Les API de réseau peuvent faire l'objet de deux types de restriction : Pour empêcher les activités nuisibles, l'accès aux ports généralement réservés est bloqué. Vous ne pouvez pas passer outre ces blocages dans votre code. Pour contrôler l'accès au réseau par le biais d'autres ports d'un fichier SWF, vous pouvez utiliser le paramètre `allowNetworking`.

Ports bloqués

A l'instar des navigateurs, Flash Player et Adobe AIR imposent des restrictions sur l'accès HTTP à certains ports. Les requêtes HTTP sont interdites sur certains ports standard traditionnellement réservés aux types de serveurs autres que HTTP.

Toute API accédant à une URL réseau est soumise aux restrictions affectant ces ports. Les API qui appellent directement des sockets, telles que `Socket.connect()` et `XMLSocket.connect()`, ou les appels à `Security.loadPolicyFile()` dans lesquels un fichier de régulation socket est en cours de chargement font exception à cette règle. Les connexions socket sont autorisées ou refusées par le biais de fichiers de régulation socket sur le serveur cible.

Les API ActionScript 3.0 concernées par le blocage des ports sont les suivantes :

```
FileReference.download(),FileReference.upload(),Loader.load(),Loader.loadBytes(),
navigateToURL(),NetConnection.call(),NetConnection.connect(),NetStream.play(),
Security.loadPolicyFile(),sendToURL(),Sound.load(),URLLoader.load(),URLStream.load()
```

Le blocage des ports s'applique également à l'importation dans une bibliothèque partagée, à l'utilisation de la balise `<img>` dans les champs de texte et au chargement de fichiers SWF sur une page HTML à l'aide des balises `<object>` et `<embed>`.

Les ports bloqués sont les suivants :

HTTP: 20 (ftp data), 21 (ftp control)

HTTP et 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)

Utilisation du paramètre allowNetworking

Vous pouvez contrôler l'accès d'un fichier SWF aux fonctionnalités réseau en définissant le paramètre `allowNetworking` dans les balises `<object>` et `<embed>` de la page HTML qui accueille le contenu SWF.

`allowNetworking` peut prendre les valeurs suivantes :

- `"all"` (par défaut) : toutes les API de réseau sont autorisées dans le fichier SWF.
- `"internal"` : le fichier SWF ne peut pas appeler les API de navigation et d'interaction avec le navigateur (présentées plus loin dans cette section). Il peut cependant appeler toutes les autres API de réseau.
- `"none"` : le fichier SWF ne peut pas appeler les API de navigation et d'interaction avec le navigateur (présentées plus loin dans cette section) ni utiliser les API de communication entre fichiers SWF (également décrites plus loin).

Le paramètre `allowNetworking` s'utilise surtout lorsque le fichier SWF et la page HTML qui l'accueille appartiennent à des domaines différents. Il est déconseillé d'utiliser la valeur `"internal"` ou `"none"` si le fichier SWF en cours de chargement appartient au même domaine que les pages HTML qui l'accueillent, car vous ne pouvez pas garantir qu'un fichier SWF est toujours chargé avec la page HTML prévue. Des personnes mal intentionnées pourraient charger un fichier SWF à partir de votre domaine sans les pages HTML correspondantes, auquel cas la restriction `allowNetworking` n'aurait pas l'effet escompté.

L'appel d'une API interdite renvoie une exception SecurityError.

Ajoutez le paramètre `allowNetworking` et définissez sa valeur dans les balises `<object>` et `<embed>` de la page HTML contenant une référence au fichier SWF, comme indiqué dans l'exemple suivant :

```
<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>
```

Une page HTML peut également utiliser un script pour générer des balises d'imbrication de fichiers SWF. Vous devez modifier le script de façon à insérer les paramètres `allowNetworking` corrects. Les pages HTML générées par Flash et Adobe Flex Builder utilisent la fonction `AC_FL_RunContent()` pour incorporer des références aux fichiers SWF. Ajoutez les paramètres `allowNetworking` au script, comme illustré ci-après :

```
AC_FL_RunContent( ... "allowNetworking", "none", ...)
```

Les API suivantes ne sont pas autorisées lorsque `allowNetworking` est défini sur `"internal"`:

`navigateToURL()`, `fscommand()`, `ExternalInterface.call()`

Outre les API recensées ci-dessus, les API ci-après sont également interdites lorsque `allowNetworking` est défini sur `"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()`

Même si le paramètre `allowNetworking` sélectionné permet au fichier SWF d'utiliser une API de réseau, d'autres restrictions peuvent survenir en fonction des limites fixées par le sandbox de sécurité (voir « Les sandbox de sécurité » à la page 716).

Si `allowNetworking` est défini sur `"none"`, vous ne pouvez pas référencer un média externe dans une balise `<img>` de la propriété `htmlText` d'un objet TextField (si vous le faites, une exception SecurityError est renvoyée).

Si `allowNetworking` est défini sur `"none"`, un symbole issu d'une bibliothèque partagée importée dans l'outil de programmation Flash (pas ActionScript) est bloqué lors de l'exécution.

Sécurité du mode plein écran

Player version 9.0.27.0 et les versions ultérieures prennent en charge le mode plein écran, dans lequel le contenu Flash Player peut remplir tout l'écran. Pour passer en mode plein écran, la propriété `displayState` de la scène est définie avec la constante `StageDisplayState.FULL_SCREEN`. Pour plus d'informations, consultez la section « Utilisation du mode plein écran » à la page 293.

L'exécution de fichiers SWF dans un navigateur impose la prise en considération de certains points de sécurité.

L'activation du mode plein écran s'effectue dans les balises `<object>` et `<embed>` de la page HTML qui contient la référence au fichier SWF. Pour ce faire, ajoutez le paramètre `allowFullScreen` et attribuez-lui la valeur `"true"` (la valeur par défaut est `"false"`), comme le montre l'exemple suivant :

```
<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>
```

Une page HTML peut également utiliser un script pour générer des balises d'imbrication de fichiers SWF. Vous devez modifier le script de manière qu'il insère les paramètres `allowFullScreen` appropriés. Les pages HTML générées par Flash et Flex Builder utilisent la fonction `AC_FL_RunContent()` pour imbriquer des références aux fichiers SWF, et vous devez ajouter le paramètre `allowFullScreen`, comme dans l'exemple suivant :

```
AC_FL_RunContent( ... "allowFullScreen", "true", ...)
```

Le code ActionScript qui lance le mode plein écran peut être uniquement appelé en réponse à un événement souris ou clavier. S'il est appelé dans d'autres situations, Flash Player renvoie une exception.

Lorsque le contenu passe en mode plein écran, un message s'affiche pour indiquer à l'utilisateur comment quitter ce mode et revenir au mode normal. Le message s'affiche pendant quelques secondes, puis s'estompe.

Pour le contenu qui s'exécute dans un navigateur, l'utilisation du clavier est restreinte en mode plein écran. Dans Flash Player 9, seuls les raccourcis clavier qui réinitialisent le mode normal de l'application (tel appuyer sur la touche Echap) sont pris en charge. Les utilisateurs ne sont pas autorisés à entrer du texte dans un champ de texte ou à naviguer à l'écran. Flash Player 10 (et versions ultérieures) prend en charge certaines touches hors impression, notamment les touches fléchées, la barre d'espacement et la touche de tabulation. La saisie de texte est néanmoins toujours interdite.

Le mode plein écran est toujours autorisé dans le lecteur autonome ou dans un fichier de projection. Par ailleurs, l'utilisation du clavier (y compris la saisie de texte) est totalement prise en charge dans ces environnements.

L'appel de la propriété `displayState` d'un objet Stage renvoie une exception pour tout appelant qui n'appartient pas au même sandbox de sécurité que le propriétaire de l'objet Stage (le fichier SWF principal). Pour plus d'informations, consultez la section « Sécurité de la scène » à la page 732.

Pour désactiver le mode plein écran pour les fichiers exécutés dans des navigateurs, les administrateurs peuvent définir `FullScreenDisable = 1` dans le fichier mms.cfg. Pour plus d'informations, consultez la section « Contrôles administrateur » à la page 718.

Pour accéder au mode plein écran dans un navigateur, un fichier SWF doit se trouver au sein d'une page HTML.

Chargement de contenu

Un fichier SWF peut charger les types de contenu suivants :

- Fichiers SWF
- Images
- Son
- Vidéo

Chargement de fichiers SWF et d'images

La classe Loader permet de charger des fichiers SWF et des images (fichiers JPG, GIF ou PNG). Un fichier SWF, s'il ne se trouve pas dans le sandbox local avec système de fichiers, peut charger des fichiers SWF et des images depuis n'importe quel domaine réseau. Seuls les fichiers SWF associés aux sandbox locaux peuvent charger des fichiers SWF et des images issus du système de fichiers local. Cependant, les fichiers du sandbox local avec réseau peuvent uniquement charger des fichiers SWF locaux qui se trouvent dans le sandbox local approuvé ou avec réseau. Les fichiers SWF associés au sandbox local avec réseau peuvent charger du contenu autre que des fichiers SWF (par exemple des images), mais ne peuvent pas accéder aux données du contenu chargé.

Lorsque vous chargez un fichier SWF d'une source non approuvée (telle qu'un domaine autre que celui du fichier SWF racine de l'objet Loader), il peut s'avérer utile de définir un masque pour ce dernier, afin d'empêcher le contenu chargé, qui est un enfant de l'objet Loader, d'apparaître dans des parties de la scène qui ne relèvent pas de ce masque, comme illustré par le code suivant :

```
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);
```

Lorsque vous appelez la méthode `load()` de l'objet Loader, vous pouvez spécifier un paramètre `context`, qui constitue un objet LoaderContext. La classe LoaderContext comporte trois propriétés qui permettent de définir le contexte d'utilisation du contenu chargé :

- `checkPolicyFile` : utilisez cette propriété uniquement pour le chargement d'un fichier image (pas pour un fichier SWF). Spécifiez-la pour un fichier image issu d'un domaine autre que celui du fichier contenant l'objet Loader. Si vous définissez cette propriété sur `true`, Loader recherche sur le serveur d'origine un fichier de régulation d'URL (voir « Contrôles de site Web (fichiers de régulation) » à la page 721). Si le serveur autorise l'accès au domaine Loader, le code ActionScript des fichiers SWF du domaine Loader peut accéder à l'image chargée. En d'autres termes, vous pouvez utiliser soit la propriété `Loader.content` pour obtenir une référence à un objet Bitmap qui représente l'image chargée, soit la méthode `BitmapData.draw()` pour accéder aux pixels de l'image chargée.
- `securityDomain` : utilisez cette propriété uniquement pour le chargement d'un fichier SWF (pas pour une image). Cette propriété peut être appelée pour un fichier SWF provenant d'un autre domaine que le fichier qui contient l'objet Loader. Seules les deux valeurs suivantes sont actuellement prises en charge par la propriété `securityDomain` : `null` (par défaut) et `SecurityDomain.currentDomain`. Si vous spécifiez `SecurityDomain.currentDomain`, le fichier SWF chargé est *importé* sur demande dans le sandbox du fichier SWF à l'origine du chargement. Par conséquent le fichier fonctionne comme s'il avait été chargé à partir du serveur du fichier appelant. Cette opération n'est permise que si le fichier de régulation d'URL se trouve sur le serveur du fichier SWF chargé, pour qu'il soit accessible au domaine du fichier SWF à l'origine du chargement. Si le fichier nécessaire est détecté, les deux fichiers peuvent librement effectuer une programmation croisée dès le début du chargement, puisqu'ils se trouvent dans le même sandbox. Notez que l'importation dans le sandbox peut presque être remplacée par un chargement ordinaire suivi d'un appel du fichier SWF chargé à la méthode `Security.allowDomain()`. Cette dernière peut s'avérer plus simple à utiliser puisque le fichier SWF chargé se trouve alors dans son sandbox naturel, pouvant ainsi accéder aux ressources de son propre serveur.
- `applicationDomain` : utilisez cette propriété uniquement lors du chargement d'un fichier SWF écrit en ActionScript 3.0 (et non une image ou un fichier SWF écrit en ActionScript 1.0 ou 2.0). Lors du chargement du fichier, vous pouvez spécifier s'il doit être placé dans un domaine d'application particulier, plutôt que dans le domaine par défaut, c'est-à-dire un nouveau domaine créé comme enfant du domaine d'application du fichier SWF à l'origine du chargement. Notez que les domaines d'application sont des sous-ensembles des domaines de sécurité. Ainsi, vous pouvez uniquement spécifier un domaine d'application cible si le fichier SWF chargé provient de votre

propre de domaine de sécurité, soit parce qu'il appartient à votre propre serveur, soit parce que vous l'avez importé dans votre domaine de sécurité à l'aide de la propriété `securityDomain`. Si vous spécifiez un domaine d'application mais que le fichier SWF chargé fait partie d'un domaine de sécurité différent, le domaine que vous spécifiez dans `applicationDomain` est ignoré. Pour plus d'informations, consultez la section « Utilisation de la classe ApplicationDomain » à la page 666.

Pour plus d'informations, consultez la section « Définition du contexte de chargement » à la page 321.

L'objet Loader possède une importante propriété, `contentLoaderInfo`, qui constitue un objet LoaderInfo. Contrairement à la plupart des objets, un objet LoaderInfo est partagé entre le fichier SWF à l'origine du chargement et le contenu chargé. Il est en outre accessible par les deux parties. Si le contenu chargé est un fichier SWF, il peut accéder à l'objet LoaderInfo au moyen de la propriété `DisplayObject.loaderInfo`. Les objets LoaderInfo incluent des informations telles que la progression du chargement, l'URL du fichier de chargement et du fichier chargé, la relation de confiance entre ces deux fichiers, et d'autres renseignements. Pour plus d'informations, consultez la section « Surveillance de la progression du chargement » à la page 320.

Chargement de sons et vidéos

En dehors des fichiers du sandbox local avec système de fichiers, tous les fichiers SWF sont autorisés à charger des sons et des vidéos en provenance d'un réseau grâce aux méthodes `Sound.load()`, `NetConnection.connect()` et `NetStream.play()`.

Seuls les fichiers SWF locaux peuvent charger des médias du système de fichiers local. Seuls les fichiers du sandbox local avec système de fichiers et du sandbox local approuvé peuvent accéder aux données de ces fichiers chargés.

D'autres restrictions s'appliquent à l'accès aux données à partir d'un média chargé. Pour plus d'informations, consultez la section « Accès aux médias chargés comme s'il s'agissait de données » à la page 734.

Chargement de fichiers SWF et d'images à l'aide de la balise <img> d'un champ de texte

La balise `<img>` permet de charger des fichiers SWF et bitmap dans un champ de texte, comme le montre le code suivant :

```
<img src = 'filename.jpg' id = 'instanceName' >
```

Pour accéder au contenu chargé de cette manière, utilisez la méthode `getImageReference()` de l'occurrence de TextField, comme dans le code suivant :

```
var loadedObject:DisplayObject = myTextField.getImageReference('instanceName');
```

Notez cependant que les fichiers SWF et images chargés de cette manière sont placés dans le sandbox correspondant à leur origine.

Lorsque vous chargez un fichier image à l'aide de la balise `<img>` d'un champ de texte, l'accès aux données de l'image peut être autorisé par le biais d'un fichier de régulation d'URL. Vous pouvez vérifier l'existence d'un tel fichier en ajoutant l'attribut `checkPolicyFile` à la balise `<img>`, comme le montre le code suivant :

```
<img src = 'filename.jpg' checkPolicyFile = 'true' id = 'instanceName' >
```

Lorsque vous chargez un SWF à l'aide de la balise `<img>` d'un champ de texte, vous pouvez autoriser l'accès aux données de ce fichier SWF via un appel à la méthode `Security.allowDomain()`.

Si vous utilisez la balise `<img>` d'un champ de texte pour charger un fichier externe (plutôt que d'incorporer une classe Bitmap dans votre fichier SWF), un objet Loader est automatiquement créé comme enfant de l'objet TextField et le fichier externe est chargé dans l'objet Loader comme si vous aviez utilisé un tel objet en ActionScript pour charger ce fichier. Dans ce cas, la méthode `getImageReference()` renvoie l'objet Loader automatiquement créé. Aucune vérification de sécurité n'est nécessaire pour charger cet objet Loader car il se trouve dans le même sandbox de sécurité que le code appelant.

Toutefois, si vous faites référence à la propriété `content` de l'objet Loader pour accéder au média chargé, des règles de sécurité s'appliquent. Si le contenu est une image, vous devez implémenter un fichier de régulation d'URL ; s'il s'agit d'un fichier SWF, vous devez modifier le code de ce fichier de manière qu'il appelle la méthode `allowDomain()`.

Contenu diffusé à l'aide de serveurs RTMP

Flash Media Server utilise le protocole RTMP (Real-Time Media Protocol) pour servir des données, des sons et des vidéos. Un fichier SWF charge ce type de média à l'aide de la méthode `connect()` de la classe NetConnection, en transmettant une URL RTMP comme paramètre. Flash Media Server peut restreindre les connexions et empêcher le téléchargement du contenu, selon le domaine du fichier requis. Pour plus d'informations, reportez-vous à la documentation Flash Media Server.

Pour les médias chargés à partir de sources RTMP, vous ne pouvez pas utiliser les méthodes `BitmapData.draw()` et `SoundMixer.computeSpectrum()` pour extraire les données image et son au moment de l'exécution.

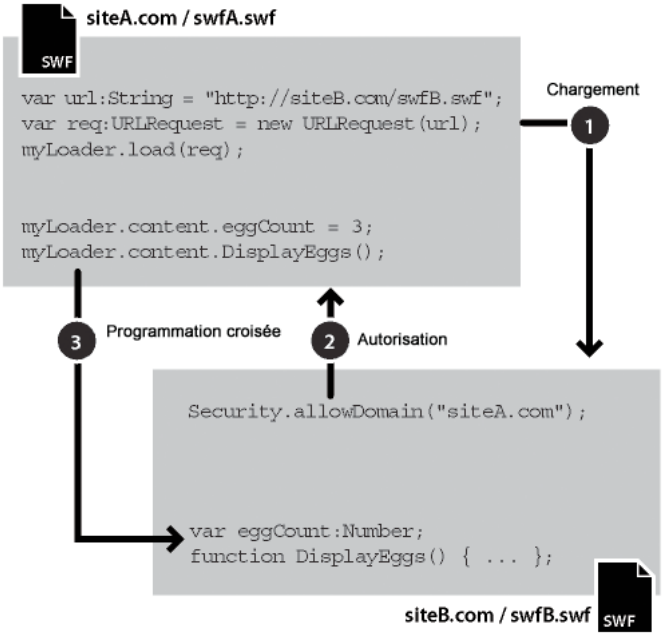
Programmation croisée

Si deux fichiers SWF écrits en ActionScript 3.0 sont servis à partir d'un même domaine (par exemple l'URL de l'un est http://www.example.com/swfA.swf et celle de l'autre est http://www.example.com/swfB.swf), alors l'un des fichiers SWF peut examiner et modifier les variables, objets, propriétés, méthodes, etc. de l'autre fichier, et inversement. On parle de *programmation croisée*.

La programmation croisée n'est pas prise en charge entre les fichiers SWF AVM1 et AVM2. Un fichier SWF AVM1 est un fichier créé avec ActionScript 1.0 ou ActionScript 2.0. (AVM1 et AVM2 font référence à la machine virtuelle ActionScript.) Vous pouvez néanmoins utiliser la classe LocalConnection pour échanger des données entre AVM1 et AVM2.

Si deux fichiers SWF écrits en ActionScript 3.0 sont servis à partir de domaines différents (par exemple http://siteA.com/swfA.swf et http://siteB.com/swfB.swf), par défaut Flash Player ne permet pas la programmation entre swfA.swf et swfB.swf, et inversement. Pour autoriser l'accès à des fichiers SWF issus d'autres domaines, un fichier SWF doit appeler `Security.allowDomain()`. Ainsi, en appelant `Security.allowDomain("siteA.com")`, swfB.swf accepte la programmation en provenance des fichiers SWF de siteA.com.

Dans tout contexte inter-domaines, il est important d'identifier clairement les parties impliquées. Dans le cadre de cette étude, le fichier effectuant la programmation croisée sera appelé *partie procédant à l'accès* (habituellement le fichier SWF procédant à l'accès), et l'autre côté sera appelé *partie cible* (généralement le fichier SWF cible). Lorsque siteA.swf programme siteB.swf, siteA.swf est la partie procédant à l'accès et site.B.swf la partie cible, comme le montre l'illustration suivante :



Les autorisations inter-domaines établies avec `Security.allowDomain()` sont asymétriques. Dans l'exemple précédent, siteA.swf peut programmer siteB.swf mais l'inverse n'est pas possible car siteA.swf n'a pas appelé la méthode `Security.allowDomain()` pour autoriser les fichiers SWF de siteB.com à le programmer. Vous pouvez définir des autorisations symétriques si les deux fichiers SWF appellent la méthode `Security.allowDomain()`.

En dehors de la protection des fichiers SWF contre les scripts inter-domaines provenant d'autres fichiers SWF, Flash Player protège également les fichiers SWF contre ce type de script provenant des fichiers HTML. La programmation HTML vers SWF est possible au moyen de rappels effectués avec la méthode `ExternalInterface.addCallback()`. Lorsque la programmation HTML vers SWF franchit les limites du domaine, le SWF cible doit également appeler `Security.allowDomain()`, comme s'il avait été appelé par un fichier SWF, faute de quoi l'opération échoue. Pour plus d'informations, consultez la section « Contrôles de création (développeur) » à la page 724.

Flash Player fournit en outre des contrôles de sécurité spécifiques à la programmation SWF vers HTML. Pour plus d'informations, consultez la section « Contrôle de l'accès URL externe » à la page 740.

Sécurité de la scène

Certaines propriétés et méthodes de l'objet Stage sont disponibles pour tout sprite ou clip de la liste d'affichage.

Le premier fichier SWF chargé est cependant considéré comme le propriétaire de l'objet Stage. Par défaut, les propriétés et méthodes suivantes de l'objet Stage sont uniquement disponibles pour les fichiers SWF du même sandbox de sécurité que le propriétaire de l'objet Stage :

Propriétés	Méthodes
align	addChild()
displayState	addChildAt()
frameRate	addEventListener()
height	dispatchEvent()
mouseChildren	hasEventListener()
numChildren	setChildIndex()
quality	willTrigger()
scaleMode	
showDefaultContextMenu	
stageFocusRect	
stageHeight	
stageWidth	
tabChildren	
textSnapshot	
width	

Pour qu'un fichier SWF d'un sandbox différent de celui du propriétaire de l'objet Stage puisse accéder à ces propriétés et méthodes, le fichier SWF propriétaire de l'objet Stage doit appeler la méthode `Security.allowDomain()`. Pour plus d'informations, consultez la section « Contrôles de création (développeur) » à la page 724.

La propriété `frameRate` est un cas à part : tout fichier SWF peut lire la propriété `frameRate`. Toutefois, seuls les fichiers situés dans le sandbox de sécurité du propriétaire de l'objet Stage (ou ceux qui ont été autorisés à l'aide de la méthode `Security.allowDomain()`) peuvent modifier cette propriété.

Il existe également des restrictions sur les méthodes `removeChildAt()` et `swapChildrenAt()`, mais ce sont des restrictions différentes des autres. Pour appeler ces méthodes, le code ne doit pas se trouver dans le même domaine que le propriétaire de l'objet Stage mais dans le même domaine que le propriétaire du ou des objets enfant concernés ; en outre, les objets enfant peuvent appeler la méthode `Security.allowDomain()`.

Parcours de la liste d'affichage

La capacité d'un fichier SWF d'accéder aux objets d'affichage chargés à partir d'autres sandbox fait l'objet de restrictions. Pour qu'un fichier SWF puisse accéder à un objet d'affichage créé par un autre fichier SWF dans un sandbox différent, le fichier SWF cible doit appeler la méthode `Security.allowDomain()` pour autoriser l'accès du domaine du fichier SWF procédant à l'appel. Pour plus d'informations, consultez la section « Contrôles de création (développeur) » à la page 724.

Pour accéder à un objet Bitmap chargé par un objet Loader, il faut qu'un fichier de régulation d'URL existe sur le serveur d'origine du fichier image et que ce fichier accorde une autorisation au domaine du fichier SWF qui essaie d'accéder à l'objet Bitmap (voir« Contrôles de site Web (fichiers de régulation) » à la page 721).

L'objet LoaderInfo qui correspond au fichier chargé (et à l'objet Loader) inclut les trois propriétés suivantes, qui définissent la relation entre l'objet chargé et l'objet Loader : `childAllowsParent`, `parentAllowsChild` et `sameDomain`.

Sécurité des événements

Les événements liés à la liste d'affichage sont soumis à des restrictions d'accès de sécurité en fonction du sandbox de l'objet d'affichage qui distribue l'événement. Un événement de la liste d'affichage traverse des phases de capture et de propagation vers le haut (voir « Gestion des événements » à la page 254). Au cours de ces deux phases un événement passe de l'objet d'affichage source aux objets d'affichage parent dans la liste d'affichage. Si un objet parent appartient à un sandbox de sécurité différent de celui de l'objet d'affichage source, la phase de capture ou de propagation vers le haut s'arrête en dessous de cet objet parent, sauf si une relation de confiance est établie entre le propriétaire de l'objet parent et celui de l'objet source. Cette confiance mutuelle s'établit des manières suivantes :

1. Le fichier SWF propriétaire de l'objet parent doit appeler la méthode `Security.allowDomain()` pour approuver le domaine du fichier SWF propriétaire de l'objet source.
2. Le fichier SWF propriétaire de l'objet source doit appeler la méthode `Security.allowDomain()` pour approuver le domaine du fichier SWF propriétaire de l'objet parent.

L'objet LoaderInfo qui correspond au fichier chargé (et à l'objet Loader) inclut les deux propriétés suivantes, qui définissent la relation entre l'objet chargé et l'objet Loader : `childAllowsParent` et `parentAllowsChild`.

Pour les événements distribués à partir d'objets autres que les objets d'affichage, il n'existe aucune vérification de sécurité ni aucune implication liée à la sécurité.

Accès aux médias chargés comme s'il s'agissait de données

Vous pouvez accéder aux données grâce aux méthodes telles que `BitmapData.draw()` et `SoundMixer.computeSpectrum()`. Par défaut, un fichier SWF d'un sandbox de sécurité ne peut obtenir de données de pixels ou de données audio de la part d'objets graphiques ou audio rendus ou lus dans un média chargé d'un autre sandbox. Vous pouvez cependant utiliser les méthodes suivantes pour accorder cette autorisation :

- Dans le fichier SWF chargé, appelez la méthode `Security.allowDomain()` pour permettre l'accès aux données des fichiers SWF d'autres domaines.
- Pour un fichier image, son ou vidéo chargé, ajoutez un fichier de régulation d'URL sur le serveur du fichier chargé. Ce fichier de régulation doit accorder l'accès au domaine du fichier SWF qui tente d'appeler la méthode `BitmapData.draw()` ou `SoundMixer.computeSpectrum()` pour extraire des données de ce fichier.

Les sections qui suivent offrent des détails sur l'accès aux données bitmap, son et vidéo.

Accès aux données bitmap

La méthode `draw()` de l'objet BitmapData vous permet d'extraire les pixels actuellement affichés de tout objet d'affichage vers l'objet BitmapData. Il peut s'agir des pixels d'un objet MovieClip, d'un objet Bitmap ou d'un objet d'affichage. Les conditions suivantes doivent être remplies pour que la méthode `draw()` puisse extraire les pixels vers l'objet BitmapData :

- Si l'objet source n'est pas un fichier bitmap chargé, l'objet source et (dans le cas d'un objet Sprite ou MovieClip) tous ses objets enfant doivent provenir du même domaine que l'objet appelant la méthode `draw()` ou se trouver dans un fichier SWF qui est devenu accessible à l'objet appelant suite à l'appel de la méthode `Security.allowDomain()`.
- Si l'objet source est un fichier bitmap chargé, cet objet doit provenir du même domaine que l'objet appelant la méthode `draw()` ou son serveur source doit inclure un fichier de régulation d'URL qui accorde l'autorisation nécessaire au domaine appelant.

Si ces conditions ne sont pas réunies, une exception SecurityError est renvoyée.

Lorsque vous appelez la méthode `load()` de la classe Loader, vous pouvez spécifier un paramètre `context`, qui constitue un objet LoaderContext. Si vous réglez la propriété `checkPolicyFile` de l'objet LoaderContext sur `true`, Flash Player recherche un fichier de régulation d'URL sur le serveur à partir duquel l'image est chargée. S'il existe un fichier de régulation autorisant le domaine du fichier SWF à l'origine du chargement, le fichier peut accéder aux données de l'objet Bitmap ; dans le cas contraire, l'accès est refusé.

Vous pouvez également spécifier une propriété `checkPolicyFile` dans une image chargée via la balise `<img>` d'un champ de texte. Pour plus d'informations, consultez la section « Chargement de fichiers SWF et d'images à l'aide de la balise <img> d'un champ de texte » à la page 730.

Accès aux données audio

Les API ActionScript 3.0 suivantes, liées aux sons, font l'objet de restrictions de sécurité :

- Méthode `SoundMixer.computeSpectrum()` : toujours autorisée pour les fichiers SWF qui se trouvent dans le même sandbox de sécurité que le fichier son. Des contrôles de sécurité sont nécessaires pour les fichiers se trouvant dans d'autres sandbox.
- Méthode `SoundMixer.stopAll()` : toujours autorisée pour les fichiers SWF qui se trouvent dans le même sandbox de sécurité que le fichier son. Des contrôles de sécurité sont nécessaires pour les fichiers se trouvant dans d'autres sandbox.
- Propriété `id3` de l'objet Sound : toujours autorisée pour les fichiers SWF qui se trouvent dans le même sandbox de sécurité que le fichier son. Des contrôles de sécurité sont nécessaires pour les fichiers se trouvant dans d'autres sandbox.

Chaque son est associé à deux types de sandbox, un sandbox de contexte et un sandbox de propriétaire :

- Le domaine d'origine du son détermine le sandbox de contexte. Celui-ci établit si les données peuvent être extraites du son via la propriété `id3` du son et la méthode `SoundMixer.computeSpectrum()`.
- L'objet qui déclenche la lecture du son détermine le sandbox de propriétaire, qui établit à son tour si le son peut être arrêté à l'aide de la méthode `SoundMixer.stopAll()`.

Lorsque vous chargez le son à l'aide de la méthode `load()` de la classe Sound, vous pouvez spécifier un paramètre `context`, qui constitue un objet SoundLoaderContext. Si vous définissez la propriété `checkPolicyFile` de l'objet SoundLoaderContext sur `true`, Flash Player recherche un fichier de régulation d'URL sur le serveur à partir duquel le son est chargé. S'il existe un fichier de régulation autorisant le domaine du fichier SWF à l'origine du chargement, le fichier peut accéder à la propriété `id` de l'objet Sound ; dans le cas contraire, l'accès est refusé. En outre, la propriété `checkPolicyFile` peut permettre d'activer la méthode `SoundMixer.computeSpectrum()` pour les sons chargés.

La méthode `SoundMixer.areSoundsInaccessible()` vous permet de savoir si l'appel à la méthode `SoundMixer.stopAll()` n'entraînerait pas l'arrêt de tous les sons parce que le sandbox de l'une ou de plusieurs des propriétés d'objet son est inaccessible à l'appelant.

La méthode `SoundMixer.stopAll()` permet d'arrêter tous les sons dont le sandbox de propriétaire est le même que celui de l'appelant de `stopAll()`. Elle arrête également les sons dont la lecture a été déclenchée par des fichiers SWF ayant appelé la méthode `Security.allowDomain()` pour autoriser le domaine du fichier SWF appelant la méthode `stopAll()`. Tous les autres sons ne sont pas arrêtés ; vous pouvez vérifier leur présence en appelant la méthode `SoundMixer.areSoundsInaccessible()`.

L'appel de la méthode `computeSpectrum()` demande que chaque son en cours de lecture soit issu du même sandbox que l'objet appelant la méthode ou de la même source qui a autorisé l'accès au sandbox de l'appelant. Autrement, une exception SecurityError est renvoyée. Pour les sons chargés à partir de sons incorporés dans la bibliothèque d'un fichier SWF, l'autorisation est accordée en appelant la méthode `Security.allowDomain()` dans le fichier SWF chargé. Pour les sons chargés à partir de sources autres que des fichiers SWF (des fichiers mp3 ou des fichiers vidéo), un fichier de régulation d'URL sur le serveur source doit autoriser l'accès aux données figurant dans le média chargé. Vous ne pouvez pas utiliser la méthode `computeSpectrum()` si le chargement du son s'effectue à partir de flux RTMP.

Pour plus d'informations, consultez les sections « Contrôles de création (développeur) » à la page 724 et « Contrôles de site Web (fichiers de régulation) » à la page 721.

Accès aux données vidéo

La méthode `BitmapData.draw()` vous permet de capturer des données de pixels à partir de l'image active d'une vidéo.

Il existe deux types de vidéo :

- La vidéo RTMP
- La vidéo progressive, chargée à partir d'un fichier FLV sans utiliser de serveur RTMP

La méthode `BitmapData.draw()` ne permet pas d'utiliser la vidéo RTMP.

Lorsque vous appelez la méthode `BitmapData.draw()` avec la vidéo progressive comme paramètre `source`, l'appelant de `BitmapData.draw()` doit provenir du même sandbox que le fichier FLV ou le serveur du fichier FLV doit contenir un fichier de régulation qui autorise le domaine du fichier SWF appelant. Pour demander le téléchargement du fichier de régulation, définissez la propriété `checkPolicyFile` de l'objet NetStream sur `true`.

Chargement des données

Les fichiers SWF permettent d'échanger des données entre un serveur et ActionScript. Le chargement de données est une opération différente du chargement de média car les informations chargées apparaissent directement dans ActionScript et non affichées sous forme de média. En règle générale, les fichiers SWF peuvent charger des données à partir de leur propre domaine. Cependant, ils ont le plus souvent besoin de fichiers de régulation pour charger des données à partir d'autres domaines (consultez la section « Contrôles de site Web (fichiers de régulation) » à la page 721).

Utilisation de URLLoader et URLStream

Vous pouvez charger des données telles que des fichiers XML ou texte. Les méthodes `load()` des classes URLLoader et URLStream sont régies par les autorisations d'un fichier de régulation d'URL.

Si vous utilisez la méthode `load()` pour charger le contenu d'un autre domaine que celui du fichier SWF qui appelle la méthode, Flash Player recherche le fichier de régulation d'URL sur le serveur des actifs chargés. S'il en existe un qui autorise l'accès au domaine du fichier SWF à l'origine du chargement, vous pouvez charger les données.

Connexion aux sockets

Par défaut, Flash Player recherche un fichier de régulation socket servi à partir du port 843. Comme dans le cas des fichiers de régulation d'URL, ce fichier est appelé *Fichier de régulation maître*.

Lors de l'introduction des fichiers de régulation dans Flash Player 6, les fichiers socket n'étaient pas pris en charge. Les connexions aux serveurs socket étaient autorisées par un fichier de régulation figurant à l'emplacement par défaut sur un serveur HTTP à condition d'utiliser le port 80 du même hôte que le serveur socket. Flash Player 9 prend toujours cette fonctionnalité en charge mais ce n'est pas le cas de Flash Player 10. Dans Flash Player 10, seuls les fichiers de régulation socket peuvent autoriser les connexions socket.

A l'instar des fichiers de régulation d'URL, les fichiers de régulation socket prennent en charge une instruction de méta-régulation qui identifie les ports pouvant servir des fichiers de régulation. La méta-régulation des fichiers de régulation socket est définie sur « all », plutôt que sur « master-only ». Par conséquent, à moins qu'un paramètre plus restrictif soit défini dans le fichier de régulation maître, Flash Player considère comme acquis que n'importe quel socket de l'hôte peut servir un fichier de régulation socket.

L'accès aux connexions socket et socket XML est désactivé par défaut, même si le socket auquel vous vous connectez se trouve dans le même domaine que le fichier SWF. Vous pouvez autoriser l'accès de niveau socket en servant un fichier de régulation socket à partir d'un des emplacements suivants :

- Le port 843 (emplacement du fichier de régulation maître)
- Le même port que la connexion socket principale
- Un autre port que la connexion socket principale

Par défaut, Flash Player recherche un fichier de régulation socket sur le port 843 et sur le même port que la connexion socket principale. Si vous souhaitez servir un fichier de régulation socket à partir d'un autre port, le fichier SWF doit appeler `Security.loadPolicyFile()`.

La syntaxe d'un fichier de régulation socket est identique à celle d'un fichier de régulation d'URL, à la différence qu'elle doit aussi spécifier les ports accessibles. Un fichier de régulation socket servi via un port dont le numéro est inférieur à 1024 peut autoriser l'accès à tous les ports. Un fichier de régulation transmis via le port 1024 ou supérieur ne peut définir l'accès qu'au port 1024 et aux ports supérieurs. Les ports accessibles sont spécifiés par l'attribut `to-ports` dans la balise `<allow-access-from>`. Il est possible d'utiliser des numéros de ports, des plages de ports et des caractères génériques.

Voici un exemple de fichier de régulation socket :

```
<?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>
```

Pour récupérer un fichier de régulation socket sur le port 843 ou sur le même port que la connexion socket principale, appelez la méthode `Socket.connect()` ou `XMLSocket.connect()`. Flash Player recherche d'abord un fichier de régulation maître sur le port 843. S'il en trouve un, il vérifie s'il contient une instruction de méta-régulation interdisant les fichiers de régulation socket sur le port cible. Si l'accès n'est pas interdit, Flash Player recherche l'instruction `allow-access-from` appropriée dans le fichier de régulation maître. En l'absence d'une telle instruction, il recherche un fichier de régulation sur le même port que la connexion socket principale.

Pour récupérer un fichier de régulation socket à un autre emplacement, appelez d'abord la méthode `Security.loadPolicyFile()` en utilisant la syntaxe `"xmlsocket"` spéciale, comme illustré ci-dessous :

```
Security.loadPolicyFile("xmlsocket://server.com:2525");
```

Appelez la méthode `Security.loadPolicyFile()` avant d'appeler `Socket.connect()` ou `XMLSocket.connect()`. Flash Player peut alors attendre le retour de votre requête de fichier de régulation avant de décider ou non d'autoriser la connexion principale. Cependant, si le fichier de régulation maître spécifie que l'emplacement cible ne peut pas servir des fichiers de régulation, l'appel à `loadPolicyFile()` est sans effet, même si un fichier de régulation se trouve à l'emplacement en question.

Si vous implémentez un serveur socket et que vous devez fournir un fichier de régulation socket, vous devez choisir entre fournir le fichier de régulation sur le port qui accepte les connexions principales et utiliser un autre port. Dans les deux cas, votre serveur doit attendre la première transmission en provenance de votre client avant d'envoyer une réponse.

Lorsque Flash Player demande un fichier de régulation, il transmet toujours la chaîne suivante dès que la connexion est établie :

```
<policy-file-request/>
```

Une fois que le serveur a reçu la chaîne, il peut transmettre le fichier de régulation. La requête émise par Flash Player se termine toujours par un octet nul et la réponse du serveur doit se terminer de même.

N'envisagez pas d'utiliser la même connexion pour la requête de fichier de régulation et pour la connexion principale. Vous devez fermer la connexion une fois le fichier de régulation transmis. A défaut, Flash Player ferme la connexion du fichier de régulation, puis établit une autre connexion pour la connexion principale.

Envoi de données

L'envoi de données s'effectue lorsque le code ActionScript issu d'un fichier SWF envoie des données à un serveur ou une ressource. L'envoi de données est toujours autorisé pour les fichiers SWF de domaines réseau. Un fichier SWF local peut envoyer des données à des adresses réseau uniquement si elles se trouvent dans le sandbox local approuvé ou local avec réseau. Pour plus d'informations, consultez la section « Les sandbox locaux » à la page 716.

Vous pouvez utiliser la fonction `flash.net.sendToURL()` pour envoyer des données à une URL. D'autres méthodes permettent d'envoyer des requêtes aux URL. Il s'agit notamment des méthodes de chargement, telles que `Loader.load()` et `Sound.load()`, et des méthodes de chargement de données, telles que `URLLoader.load()` et `URLStream.load()`.

Chargement et téléchargement de fichiers

La méthode `FileReference.upload()` lance le chargement d'un fichier sélectionné par l'utilisateur vers un serveur distant. Vous devez appeler la méthode `FileReference.browse()` ou `FileReferenceList.browse()` avant la méthode `FileReference.upload()`.

Le code ActionScript qui lance la méthode `FileReference.browse()` ou `FileReferenceList.browse()` ne peut être appelé qu'en réponse à un événement souris ou clavier. S'il est appelé dans d'autres situations, Flash Player 10 et ultérieur renvoie une exception.

L'appel de la méthode `FileReference.download()` ouvre une boîte de dialogue dans laquelle l'utilisateur peut télécharger un fichier à partir d'un serveur distant.

Remarque : *si votre serveur nécessite une authentification des utilisateurs, seuls les fichiers SWF s'exécutant dans un navigateur (c'est-à-dire utilisant le plug-in du navigateur ou un contrôle ActiveX) peuvent fournir une boîte de dialogue pour demander à l'utilisateur un nom et un mot de passe d'authentification, ceci uniquement pour les téléchargements. Flash Player ne permet pas de charger des fichiers sur un serveur qui nécessite une authentification utilisateur.*

Les chargements et téléchargements ne sont autorisés que si le fichier SWF appelant appartient au sandbox local avec système de fichiers.

Par défaut, un fichier SWF ne peut pas lancer un chargement ou un téléchargement avec un serveur autre que le sien. Il peut le faire si le serveur en question fournit un fichier de régulation accordant un accès au domaine du fichier SWF appelant.

Chargement de contenu incorporé à partir de fichiers SWF importés dans un domaine de sécurité

Lorsque vous chargez un fichier SWF, vous pouvez définir le paramètre `context` de la méthode `load()` de l'objet Loader utilisé pour le chargement. Ce paramètre prend un objet LoaderContext. Si vous réglez la propriété `securityDomain` de cet objet LoaderContext sur `Security.currentDomain`, Flash Player recherche un fichier de régulation d'URL sur le serveur à partir duquel le fichier SWF est chargé. S'il en existe un qui autorise l'accès au domaine du fichier SWF à l'origine du chargement, vous pouvez charger le fichier SWF sous forme de média importé. De cette manière, le fichier à l'origine du chargement obtient l'accès aux objets de la bibliothèque du fichier SWF.

Une autre méthode permet d'autoriser l'accès d'un fichier SWF aux classes des fichiers SWF chargés à partir d'un autre sandbox de sécurité : le fichier SWF chargé doit simplement appeler la méthode `Security.allowDomain()` pour autoriser l'accès du domaine du fichier SWF appelant. Cet appel à la méthode `Security.allowDomain()` peut s'ajouter à la méthode constructeur de la classe principale du fichier SWF chargé. Ensuite, le fichier SWF à l'origine du chargement doit ajouter un écouteur d'événement pour répondre à l'événement `init` distribué par la propriété `contentLoaderInfo` de l'objet Loader. Cet événement est distribué lorsque le fichier SWF chargé a appelé la méthode `Security.allowDomain()` dans la méthode constructeur et que des classes du fichier SWF chargé sont disponibles pour le fichier SWF à l'origine du chargement. Ce dernier peut alors extraire les classes du fichier SWF chargé en appelant `Loader.contentLoaderInfo.applicationDomain.getDefinition()`.

Utilisation de contenus existants

Dans Flash Player 6, le domaine utilisé par certains paramètres Flash Player dépend de la fin du domaine du fichier SWF. Il s'agit notamment des paramètres d'autorisations relatifs à la caméra et au microphone, aux quotas de stockage et au stockage d'objets partagés persistants.

Si le domaine du fichier SWF comprend plus de deux segments, par exemple www.example.com, le premier segment du domaine (www) est supprimé et la fin du domaine est exploitée. Ainsi, dans Flash Player 6, www.exemple.com et magasin.exemple.com ont en commun le domaine « example.com » pour ces paramètres. De même, www.exemple.co.fr et magasin.exemple.co.fr ont tous les deux recours au domaine exemple.co.fr pour ces paramètres. Cette caractéristique pose problème pour les fichiers SWF issus de domaines distincts, tels que example1.co.uk et example2.co.uk, qui ont alors accès aux mêmes objets partagés.

A compter de Flash Player 7, les paramètres du lecteur sont choisis par défaut en fonction du domaine exact d'un fichier SWF. Par exemple, le fichier SWF de www.exemple.com applique les paramètres du lecteur de www.exemple.com, et le fichier SWF de magasin.exemple.com utiliserait les paramètres différents de magasin.exemple.com.

Dans le cas d'un fichier SWF écrit en ActionScript 3.0, si `Security.exactSettings` conserve la valeur par défaut `true`, Flash Player utilise les domaines exacts pour les paramètres de lecteur. Si sa valeur est `false`, Flash Player utilise les paramètres de domaine de Flash Player 6. Si vous modifiez la valeur de `exactSettings`, vous devez le faire avant que ne survienne tout événement obligeant Flash Player à choisir des paramètres de lecteur (par exemple l'utilisation d'une caméra ou d'un microphone, ou l'extraction d'un objet partagé persistant).

Si vous avez publié un fichier SWF avec la version 6 et créé des objets partagés persistants à partir de ce fichier, vous devez, pour récupérer ces objets persistants à partir d'un fichier SWF en ActionScript 3.0, attribuer la valeur `false` à `Security.exactSettings` avant d'appeler `SharedObject.getLocal()`.

Définition des autorisations LocalConnection

La classe LocalConnection permet de développer des fichiers SWF capables de s'échanger des instructions. Les objets LocalConnection peuvent communiquer uniquement avec les fichiers SWF s'exécutant sur le même ordinateur client, mais peuvent s'exécuter dans diverses applications, par exemple un fichier SWF s'exécutant dans un navigateur et un fichier SWF s'exécutant dans une projection.

Chaque communication LocalConnection implique un fichier SWF émetteur et un fichier SWF récepteur. Par défaut, Flash Player permet les communications LocalConnection entre les fichiers SWF d'un même domaine. Pour les fichiers SWF de sandbox différents, le récepteur doit accorder une autorisation à l'émetteur à l'aide de la méthode `LocalConnection.allowDomain()`. La chaîne passée comme argument à la méthode `LocalConnection.allowDomain()` peut contenir n'importe lesquels des éléments suivants : noms de domaine exacts, adresses IP et caractère générique `*`.

Le format de la méthode `allowDomain()` n'est plus le même que dans ActionScript 1.0 et 2.0. Dans ces versions, `allowDomain` était une méthode de rappel que vous implémentiez. Dans ActionScript 3.0, `allowDomain()` est une méthode intégrée de la classe LocalConnection que vous appelez. Le fonctionnement de la nouvelle version de `allowDomain()` est semblable à celui de `Security.allowDomain()`.

Un fichier SWF peut utiliser la propriété `domain` de la classe LocalConnection pour déterminer le domaine.

Contrôle de l'accès URL externe

Les API ActionScript 3.0 suivantes permettent de contrôler la programmation et l'accès URL externes (via l'utilisation d'URL HTTP, de mailto:, etc.) :

- La fonction `flash.system.fscommand()`
- La méthode `ExternalInterface.call()`
- La fonction `flash.net.navigateToURL()`

Pour les fichiers SWF exécutés localement, les appels à ces méthodes aboutissent uniquement si le fichier SWF et la page Web qui le contient (le cas échéant) se trouvent dans le même sandbox de sécurité. Ils échouent si le contenu provient du sandbox local avec réseau ou local avec système de fichiers.

Pour les fichiers SWF qui ne sont pas exécutés localement, toutes ces API peuvent communiquer avec la page Web à laquelle elles sont intégrées, selon la valeur du paramètre AllowScriptAccess décrit ci-dessous. La fonction `flash.net.navigateToURL()` peut en outre communiquer avec toute fenêtre ou tout cadre de navigateur ouvert, pas seulement avec la page contenant le fichier SWF. Pour plus d'informations à ce sujet, consultez la section « Utilisation de la fonction navigateToURL() » à la page 741.

Le paramètre `AllowScriptAccess` du code HTML qui charge un fichier SWF permet de contrôler l'accès URL externe à partir du fichier SWF. Définissez ce paramètre dans la balise PARAM ou EMBED. Si la valeur du paramètre `AllowScriptAccess` n'est pas définie, le fichier SWF et la page HTML peuvent uniquement communiquer s'ils appartiennent au même domaine.

Le paramètre `AllowScriptAccess` prend en charge trois valeurs : `"always"`, `"sameDomain"` et `"never"`.

- Lorsque le paramètre `AllowScriptAccess` est défini sur `"always"`, le fichier SWF peut communiquer avec la page HTML à laquelle il est intégré, même s'il ne provient pas du même domaine qu'elle.
- Lorsque le paramètre `AllowScriptAccess` est défini sur `"sameDomain"`, le fichier SWF peut communiquer avec la page HTML à laquelle il est intégré, uniquement s'il provient du même domaine. Il s'agit de la valeur par défaut du paramètre `AllowScriptAccess`. Utilisez ce réglage ou ne définissez pas la valeur du paramètre `AllowScriptAccess` pour empêcher un fichier SWF d'un domaine d'accéder à un script sur une page HTLM issue d'un autre domaine.
- Lorsque le paramètre `AllowScriptAccess` est défini sur `"never"`, le fichier SWF ne peut communiquer avec aucune page HTML. Il est déconseillé d'utiliser cette valeur dans Adobe Flash CS4 Professional. Elle n'est pas recommandée et elle ne devrait pas s'avérer nécessaire si vous ne servez pas de fichiers SWF non approuvés à partir de votre propre domaine. Si vous devez envoyer des fichiers SWF non approuvés, Adobe vous conseille de créer un sous-domaine distinct et d'y placer l'ensemble du contenu non approuvé.

L'exemple suivant illustre le réglage de la balise `AllowScriptAccess` dans une page HTML pour autoriser l'accès URL externe à un autre domaine :

```
<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>
```

Utilisation de la fonction navigateToURL()

Outre le réglage de sécurité spécifié par le paramètre `allowScriptAccess` (voir plus haut), la fonction `navigateToURL()` propose un autre paramètre facultatif : `target`. Le paramètre `target` permet de spécifier le nom d'une fenêtre ou d'un cadre HTML auquel envoyer la requête URL. D'autres restrictions de sécurité s'appliquent à de telles requêtes et elles varient selon que la fonction `navigateToURL()` est utilisée comme instruction de programmation ou non.

Pour les instructions de programmation, telles que `navigateToURL("javascript: alert('Hello from Flash Player.')")`, les règles suivantes s'appliquent :

- Si le fichier SWF est un fichier local approuvé, la requête aboutit.
- Si la cible est la page HTML à laquelle le fichier SWF est intégré, les règles `allowScriptAccess` décrites plus haut s'appliquent.
- Si la cible contient du contenu issu du même domaine que le fichier SWF, la requête aboutit.
- Si la cible contient du contenu issu d'un autre domaine que le fichier SWF et qu'aucune des deux conditions précédentes n'est remplie, la requête échoue.

Pour les instructions autres que les instructions de programmation (telles que HTTP, HTTPS et `mailto:`), la requête échoue si toutes les conditions suivantes sont réunies :

- La cible correspond à un des mots clés spéciaux `"_top"` ou `"_parent"`
- Le fichier SWF se trouve dans une page Web hébergée dans un domaine différent et
- Le paramètre `allowScriptAccess` n'est pas défini sur `"always"` dans le fichier SWF

Pour plus d'informations

Pour plus d'informations sur l'accès URL externe, consultez les rubriques suivantes du *Guide de référence du langage et des composants ActionScript 3.0* :

- La fonction `flash.system.fscommand()`
- La méthode `call()` de la classe ExternalInterface
- La fonction `flash.net.navigateToURL()`

Objets partagés

Flash Player offre la possibilité d'utiliser des *objets partagés*, c'est-à-dire des objets ActionScript qui persistent en dehors d'un fichier SWF, soit localement dans le système de fichiers de l'utilisateur, soit à distance sur un serveur RTMP. Les objets partagés, comme d'autres médias dans Flash Player, sont répartis dans des sandbox de sécurité. Le modèle de sandbox des objets partagés s'avère toutefois différent parce que les objets partagés ne sont pas des ressources accessibles depuis d'autres domaines. Les objets partagés sont au contraire extraits d'un magasin d'objets partagés propre au domaine de chaque fichier SWF qui appelle les méthodes de la classe SharedObject. En règle générale, un magasin d'objets partagés est encore plus spécifique que le domaine d'un fichier SWF : par défaut, chaque fichier SWF utilise un magasin d'objets partagés propre à la totalité de son URL d'origine.

Le fichier SWF peut utiliser le paramètre `localPath` des méthodes `SharedObject.getLocal()` et `SharedObject.getRemote()` afin d'exploiter un magasin d'objets partagés associé à une partie de son URL seulement. Ainsi, le fichier SWF peut autoriser le partage avec d'autres fichiers SWF issus d'autres URL. Même si la valeur transmise au paramètre `localPath` est `'/'` , celle-ci spécifie tout de même un magasin d'objets partagés propres à son domaine.

Les utilisateurs peuvent limiter l'accès aux objets partagés via la boîte de dialogue Paramètres de Flash Player ou via le Gestionnaire des paramètres. Par défaut, le volume d'objets partagés créés ne peut dépasser 100 Ko de données par domaine. Les administrateurs et les utilisateurs peuvent également limiter la capacité à écrire dans le système de fichiers. Pour plus d'informations, consultez les sections « Contrôles administrateur » à la page 718 et « Contrôles utilisateur » à la page 720.

Vous pouvez définir la sécurisation d'un objet partagé en attribuant la valeur `true` au paramètre `secure` de la méthode `SharedObject.getLocal()` ou `SharedObject.getRemote()`. Notez les points suivants concernant le paramètre `secure` :

- Si ce paramètre est défini sur `true`, Flash Player crée un nouvel objet partagé sécurisé ou obtient une référence à un objet partagé sécurisé existant. Cet objet sécurisé partagé peut uniquement être lu par des fichiers SWF ou écrit dans des fichiers SWF reçus via HTTPS appelant `SharedObject.getLocal()` avec le paramètre `secure` défini sur `true`.
- Si ce paramètre est défini sur `false`, Flash Player crée un objet partagé ou obtient une référence à un objet partagé existant, qui peut être lu ou écrit par des fichiers SWF reçus via des connexions autres que HTTPS.

Si le fichier SWF appelant ne provient pas d'une URL HTTPS, l'attribution de la valeur `true` au paramètre `secure` de la méthode `SharedObject.getLocal()` ou `SharedObject.getRemote()` renvoie une exception SecurityError.

Le choix du magasin d'objets partagé dépend de l'URL d'origine du fichier SWF. Cela reste vrai même dans les deux cas de figure où un fichier SWF ne provient pas d'une URL simple : importation et chargement dynamique. L'importation s'applique lorsque vous chargez un fichier SWF dont la propriété `LoaderContext.securityDomain` a la valeur `SecurityDomain.currentDomain`. Dans ce cas, le fichier SWF chargé portera une pseudo-URL qui commence par le domaine du fichier SWF à l'origine du chargement, puis spécifie sa véritable URL d'origine. Le chargement dynamique renvoie au chargement d'un fichier SWF à l'aide de la méthode `Loader.loadBytes()`. Dans

cette situation, le fichier SWF chargé porte une pseudo-URL qui commence par l'URL complète du fichier SWF à l'origine du chargement, suivie d'un entier d'identification. Dans les deux cas de figure (importation et chargement dynamique), la pseudo-URL du fichier SWF peut être analysée à l'aide de la propriété `LoaderInfo.url`. Cette pseudo-URL est traitée comme une URL réelle pour définir le magasin d'objets partagés. Elle peut être utilisée en partie ou dans son intégralité comme paramètre `localPath` d'un objet partagé.

Les utilisateurs et les administrateurs peuvent choisir de désactiver l'utilisation d'*objets partagés tiers*. Il s'agit des objets partagés utilisés par tout fichier SWF exécuté dans un navigateur Web lorsque l'URL d'origine de ce fichier SWF est d'un domaine différent de l'URL affichée dans la barre d'adresse du navigateur. Les administrateurs et utilisateurs peuvent choisir de désactiver l'utilisation des objets partagés tiers pour des raisons de confidentialité, s'ils souhaitent éviter la surveillance inter-domaines. Pour éviter cette restriction, il est judicieux de veiller à ce que tous les fichiers SWF utilisant des objets partagés soient chargés uniquement dans une structure de pages HTML qui garantit que le fichier provient du même domaine que celui affiché dans la barre d'adresse du navigateur. Si vous essayez d'utiliser des objets partagés à partir d'un fichier SWF tiers et que l'utilisation des objets partagés tiers est désactivée, les méthodes `SharedObject.getLocal()` et `SharedObject.getRemote()` renvoient la valeur `null`. Pour plus d'informations, visitez les adresses www.adobe.com/products/flashplayer/articles/thirdpartylso.

Accès à la caméra, au microphone, au presse-papiers, à la souris et au clavier

Lorsqu'un fichier SWF essaie d'accéder à la caméra ou au microphone de l'utilisateur à l'aide de la méthode `Camera.get()` ou `Microphone.get()`, Flash Player affiche une boîte de dialogue de confidentialité, dans laquelle l'utilisateur peut autoriser ou refuser l'accès à sa caméra ou son microphone. L'utilisateur et l'administrateur peuvent également désactiver l'accès à la caméra de manière globale ou pour chaque site, grâce aux commandes du fichier mms.cfg, de l'interface de paramétrage et du Gestionnaire des paramètres (voir « Contrôles administrateur » à la page 718 et « Contrôles utilisateur » à la page 720). Si des restrictions utilisateur s'appliquent, les méthodes `Camera.get()` et `Microphone.get()` renvoient chacune la valeur `null`. La propriété `Capabilities.avHardwareDisable` vous permet de déterminer si l'administrateur a interdit (valeur `true`) ou autorisé (valeur `false`) la caméra et le microphone.

La méthode `System.setClipboard()` autorise un fichier SWF à remplacer le contenu du presse-papiers par une chaîne de caractères en texte brut, ce qui ne pose aucun risque de sécurité. Pour éviter que les mots de passe et autres données sensibles ne soient coupés ou copiés dans le Presse-papiers, il n'existe pas de méthode `getClipboard()` correspondante.

Une application qui s'exécute dans Flash Player peut uniquement contrôler les événements de clavier et de souris qui se produisent dans son focus. un contenu qui s'exécute dans Flash Player ne peut pas détecter d'événements de clavier ou de souris dans une autre application.

Index

B

C

D

E

F

J

L

P

Q

R

S

T

W

X

Z