

# Flex 4 en un jour

Ce document est une traduction d'un mini livre nommé [Flex 4 in a Day](#) de Mihai Corlan. La traduction a été réalisée par Antony Chauviré – <http://www.bases-as3.fr>

Flex 4 en un jour .....	1
A propos de ce document .....	2
Introduction .....	2
MXML 2009.....	2
Namespaces.....	2
Etats .....	3
Nouvelles balises MXML .....	4
Declarations .....	4
Library .....	5
Definition .....	5
Private .....	5
Reparent .....	6
Design Layer.....	6
Liaison de données bidirectionnelle .....	6
Dessins MXML and FXG .....	8
Dessins MXML .....	8
FXG .....	9
CSS.....	11
Thèmes Flex .....	13
Création d'un thème Flex .....	14
Appliquer un thème .....	15
Composants Spark .....	16
Le contrat de personnalisation – Personnalisation des composants Spark.....	18
Les données.....	19
Les parties (parts) .....	19
Les états.....	19
Création de composants et de skins .....	20
Cycle de vie de composants et de skins .....	22
Le nouveau moteur de mise en page .....	24
Les mises en page Spark existantes .....	24
BasicLayout.....	24
HorizontalLayout.....	25
TileLayout .....	25
VerticalLayout .....	25
Création d'une mise en page Spark personnalisée .....	26
Le nouveau moteur d'animation .....	28
Construire des effets.....	28
Effets avancés .....	29
Les nouveautés d'Adobe AIR 2 .....	32
Processus natifs .....	32
Mise à jour de version de WebKit.....	32
Nouvelle API réseau .....	33
Meilleure intégration avec le système d'exploitation .....	34
L'enregistrement audio local.....	34
Support de la gestion des gestes et du touché multiple .....	34
Le framework Text Layout .....	36
Utilisation des polices .....	36
Travailler avec des composants Flex 3 et 4 dans le même projet .....	38
Migrer des projets Flex 3 vers Flex 4.....	39
Nouvelles fonctionnalités de Flash Builder 4 (éditeur d'états, génération de code pour les accesseurs d'attributs, les écouteurs d'événements, gestion des performances et gestion du débogage) .....	40
Ressources .....	41

## ***A propos de ce document***

Bien qu'il soit impossible de couvrir tous les aspects d'un framework ou d'un langage en seulement 40 pages, nous sommes sûrs qu'en lisant ce document, vous réussirez à trouver votre chemin quand vous traiterez vos projets Flex. Vous trouverez sur le web de nombreux articles intéressants et une large documentation sur Flex sur le site Adobe. Toutefois, ce document tente concentrer l'essentiel et de vous donner assez d'informations pour avoir une bonne vue d'ensemble et savoir où en chercher d'autres.

A propos des auteurs:

Mihai Corlan est évangéliste Adobe autour de la Flash Platform. Ces 2 dernières années, il a voyagé et animé des conférences ou dialogué avec des partenaires / clients à travers le monde. Il met l'accent sur Flex, AIR et Flex Builder. Vous pouvez le suivre sur son blog <http://corlan.org> ou sur twitter <http://twitter.com/mcorlan>

Alin Achim est chercheur en informatique pour Adobe Systems. Il fait partie d'une équipe en charge de concevoir des logiciels utilisés par les employés d'Adobe. Flex (en combinaison avec diverses technologies côté serveur) représente l'épine dorsale pour les solutions qu'ils construisent.

## ***Introduction***

Si vous pensez à Flex 3 et Flex 2, vous pouvez dire que la transition était simple et facile à suivre. Comparé à cela, Flex 4 est une révolution sanglante: les choses existantes ont été modifiées et tout un tas de nouvelles fonctionnalités ont été ajoutées.

Les nouvelles fonctionnalités les plus notables sont:

- Une nouvelle architecture de composants d'interface nommée Spark qui travaille en combinaison avec l'ancienne, nommée Halo. Cette architecture sépare la mise en forme et la mise en page des données et comportements.
- Nouveau moteur d'animation
- L'ajout des graphiques FXG et MXML
- Nouvelles fonctionnalités du langage MXML
- L'ajout du support du Text Layout Framework sur les contrôles de texte
- Les graphiques de données, le contrôle AdvancedDataGrid, le contrôle OLAPDataGrid font partie du SDK Flex open source

## ***MXML 2009***

En raison des changements dans le langage MXML, un nouveau namespace a été créé. Dans ce chapitre, vous allez découvrir les nouvelles balises du langage MXML 2009 et les différents namespaces disponibles.

## ***Namespaces***

Le premier gros changement dans Flex 4 est la séparation du namespace pour le langage MXML du namespace pour la librairie de composants. Ainsi, maintenant vous spécifiez le namespace pour le langage MXML (2009 ou 2006) et pour la librairie de composants Spark, et pour la librairie de

composants Halo si vous souhaitez utiliser une ou plusieurs de ces bibliothèques. Cette séparation est nécessaire parce qu'il y a des composants avec le même nom dans les deux bibliothèques (par exemple Button, Label, etc)

Vous pouvez toujours utiliser le namespace 2006, mais ne pourrez pas utiliser des fonctionnalités de Flex 4 (comme les nouveaux composants, le moteur d'états, ou les modes de mise en page) avec lui. Cependant, vous ne pouvez pas mélanger les namespaces du langage dans le même document, il est possible d'avoir les documents qui utilisent MXML 2009 et autres documents qui utilisent MXML 2006.

Voici les principaux namespaces de Flex 4:

- **Declaration langage MXML**  
`xmlns:fx="http://ns.adobe.com/mxml/2009"`
- **Composants Halo (incluant tous les composants dans les packages mx.\*, les composants de graphiques de données, et les composants de visualisation de données):**  
`mxml>xmlns:mx="library://ns.adobe.com/flex/mx"`
- **Composants Spark (incluant tous les composants dans les packages spark.\* et les classes du framework de texte dans les packages flashx.\*):**  
`mxml>xmlns:s="library://ns.adobe.com/flex/spark"`
- **Les anciennes bibliothèques et composants Flex 3 (quand vous ne voulez pas utiliser les nouveautés du langage ou les composants Spark):**  
`xmlns:mx="http://www.adobe.com/2006/mxml"`

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               xmlns:s="library://ns.adobe.com/flex/spark">
  <fx:Style></fx:Style>
  <s:Button label="This is a Spark button!"/>
  <mx:Button label="This is a Halo button!"
/>
</s:Application>
```

Si vous travaillez avec Flash Catalyst ou Adobe Illustrator / Photoshop pour créer des skins ou des applications Flex, vous pourriez voir d'autres espaces tels que:

```
xmlns:ai="http://ns.adobe.com/ai/2009"
xmlns:d="http://ns.adobe.com/fxg/2008/dt"
xmlns:flm="http://ns.adobe.com/flame/2008"
```

Ces namespaces sont soit liés à FXG ou utilisé en interne par Flash Catalyst, Flash Builder et Illustrator ou Photoshop.

## Etats

Dans Flex 4, les états ont été fortement modifiés afin de rendre leur utilisation beaucoup plus simple. Si vous souhaitez utiliser les états dans un composant ou d'un document MXML vous utilisez un tag `<s:States>` comme un enfant direct de la balise racine afin de déclarer un certain nombre d'états, et puis vous pouvez utiliser les attributs `includeIn` / `excludeFrom` sur n'importe quel élément pour contrôler si le composant est présent dans un état donné. Vous pouvez changer l'état en utilisant la même propriété `currentState`.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx">
```

```

<s:states>
  <s:State name="default"/>
  <s:State name="report"/>
  <s:State name="form"/>
</s:states>
<s:CheckBox label="Checkbox" includeIn="report, form"/>
<s:TextInput excludeFrom="form"/>

```

En outre, vous pouvez définir la valeur pour tout attribut d'un composant MXML par suffixation avec l'attribut `.state-name`:

```

<s:CheckBox label.default = "Checkbox"
            label.report="Report Name: "
            label="" />

```

Une autre nouveauté est l'introduction de l'attribut `stateGroups` qui vous permet de regrouper et d'inclure un composant dans plusieurs états. Par exemple, regardez le code ci-dessous:

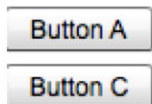
```

<s:states>
  <s:State name="A" stateGroups="G1"/>
  <s:State name="B" stateGroups="G2"/>
  <s:State name="C" stateGroups="G1"/>
</s:states>
...
<s:Button label="Button A" includeIn="G1"/>
<s:Button label="Button B" includeIn="G2"/>
<s:Button label="Button C" includeIn="G1"/>

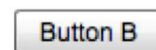
```

et son résultat:

When currentState = A or C



When currentState = B



## Nouvelles balises MXML

En plus des balises existantes (Binding, Component, Metadata, Model, Script, Style, XML, XMLList) il y a 6 nouvelles balises: Declarations, Definition, DesignLayer, Library, Reparent, and Private.

### Declarations

Dans Flex 4, tous les composants déclarés qui ne sont pas des composants d'interfaces doivent être placés à l'intérieur d'une balise Declarations:

```

<?xml version="1.0"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
              xmlns:mx="library://ns.adobe.com/flex/mx"
              xmlns:s="library://ns.adobe.com/flex/spark">
  <fx:Declarations>
    <s:HTTPService id="myService"/>
  </fx:Declarations>

```

```
        <s:Button label="My Button" />
</s:Application>
```

Si vous ne le faites pas, une erreur de compilation sera soulevée.

## Library

Utilisez la balise `<fx:Library>` pour définir, en tant qu'enfant, un ou plusieurs dessins nommés `<fx:Definition>`. La définition, elle-même dans une bibliothèque, n'est pas une instance de ce graphique, mais il vous permet de référencer cette définition un certain nombre de fois dans le document comme une instance.

La balise Library doit être le premier enfant de la balise racine du document. Vous ne pouvez avoir qu'une seule balise Library par document.

## Definition

Vous utilisez la balise `<fx:Definition>` à l'intérieur de la balise `<fx:Library>` quand vous voulez déclarer un composant d'interface à l'intérieur d'un fichier MXML, et que vous voulez l'utiliser n'importe où dans le fichier. C'est l'équivalent de définir le composant dans un fichier MXML ou ActionScript séparé.

Chaque définition crée doit avoir un attribut name défini. Il est utilisé afin d'ajouter le composant dans votre code. Le fait que vous définissiez un composant à l'aide de la balise de définition ne signifie pas que le composant sera instancié et ajouté à la liste d'affichage.

```
<?xml version="1.0"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               xmlns:s="library://ns.adobe.com/flex/spark">
    <fx:Library>
        <fx:Definition name="MySquare">
            <s:Group>
                <s:Rect width="100%" height="100%">
                    <s:stroke>
                        <s:SolidColorStroke color="red"/>
                    </s:stroke>
                </s:Rect>
            </s:Group>
        </fx:Definition>
    </fx:Library>
    <mx:Canvas>
        <fx:MySquare x="0" y="0" height="20" width="20"/>
        <fx:MySquare x="25" y="0" height="20" width="20"/>
    </mx:Canvas>
</s:Application>
```

Chaque définition dans la balise Library est compilée dans une classe ActionScript distincte qui est une sous-classe du type représenté par le premier nœud dans la définition. Dans l'exemple précédent, la nouvelle classe est une sous-classe de `mx.graphics.Group`. La portée de cette classe est limitée au document. Elle devrait être traitée comme une classe ActionScript privée.

## Private

La balise `<fx:Private>` fournit des informations sur les documents MXML ou FXG. La balise doit être un enfant de la balise racine et doit être la dernière balise du fichier. Le compilateur ignore tout le contenu de la balise `<fx:Private>`, cependant le contenu doit être valide XML. Le XML peut-être vide, peut contenir d'autres balises arbitraires, ou une chaîne de caractères.

```
<?xml version="1.0"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               xmlns:s="library://ns.adobe.com/flex/spark">
  <mx:Canvas>
    <fx:MySquare x="0" y="0" height="20" width="20"/>
    <fx:MySquare x="25" y="0" height="20" width="20"/>
  </mx:Canvas>
  <fx:Private>
    <fx:Date>10/22/2008</fx:Date>
    <fx:Author>Nick Danger</fx:Author>
  </fx:Private>
</s:Application>
```

## Reparent

Vous utilisez la balise `<fx:Reparent>` pour changer le conteneur parent d'un composant lors d'un changement d'état d'affichage:

```
<fx:Reparent target="targetComp" includeIn="stateName">
```

Target spécifie le composant cible, et `includeIn` spécifie l'état d'affichage. Quand l'état d'affichage courant est défini sur `stateName`, le composant cible devient un enfant du conteneur parent de la balise `<fx:Reparent>`.

```
<s:states>
  <s:State name="Parent1"/>
  <s:State name="Parent2"/>
</s:states>
<mx:HDividedBox height="25%" width="100%" borderStyle="inset">
  <mx:VBox id="VB1">
    <mx:Button id="setCB" includeIn="Parent1"/> </mx:VBox>
    <mx:VBox id="VB2">
      <fx:Reparent target="setCB" includeIn="Parent2"/> </mx:VBox>
  </mx:HDividedBox>
```

## Design Layer

La balise `<fx:DesignLayer>` est réservée à un usage interne. Par exemple, Flash Catalyst utilise cette balise.

## Liaison de données bidirectionnelle

Quand vous liez une variable à une propriété d'un objet et qu'à chaque fois que la variable est mise à jour, l'objet met automatiquement à jour sa propriété et qu'à chaque fois que la propriété est mise à jour, la variable est automatiquement mise à jour, vous avez ce que l'on appelle une liaison de données bidirectionnelle.

Avec Flex 3, une liaison de données bidirectionnelle était possible en utilisant une combinaison d'accolades, d'instructions `<mx:Binding>`, et des appels à la méthode `mx.binding.utils.BindingUtils.bindProperty()`.

Flex 4 introduit quelques raccourcis pour accomplir cette tâche. Les deux façons de spécifier une liaison de données bidirectionnelle sont:

- Déclaration en ligne en utilisant la syntaxe `@{bindable_property}`  
`<s:TextInput id="t1" text="@{t2.text}"/>`  
`<s:TextInput id="t2"/>`
- Dans le MXML  
`<fx:Binding source="a.property" destination="b.property" twoWay="true"/>`

## Dessins MXML and FXG

Un des plus grands changements apportés par Flex 4 est l'ajout de dessins MXML et le support du langage FXG. Bien que ces deux concepts permettent de dessiner des graphiques vectoriels, ils ne sont pas les mêmes.

### Dessins MXML

Les dessins MXML sont une collection de classes que vous utilisez pour définir des dessins interactifs (ce sont des dessins qui peuvent changer à l'exécution). Il ne s'agit pas d'un nouveau langage, mais d'un ajout au langage MXML et au framework Flex. Ces classes font parties du SDK Flex (beaucoup d'entre elles font parties des packages *mx.graphics* et *spark.primitives*) et le compilateur MXML relie chaque balise de dessins MXML vers une classe ActionScript correspondante.

Les balises de dessins MXML peuvent être ajoutées en tant qu'enfant de la balise racine Application ou de n'importe quel conteneur ou groupe. Elles ont un ordre de profondeur implicite: chaque balise est rendue au-dessus de la précédente. Parce que chaque balise correspond à une classe ActionScript, vous pouvez créer des dessins en utilisant ActionScript au lieu du MXML.

Les dessins MXML définissent:

- Des primitives de dessins et de textes
- Des remplissages, traits, dégradés, et images
- Des filtres, masques, opacités, transformations et modes de fusion

Voici une liste de balises de dessins MXML et leurs implémentations ActionScript (veuilles noter que le namespace s est pour la librairie de composants Spark):

<s:BitmapFill>	mx.graphics.BitmapFill
<s:BitmapImage>	spark.primitives.BitmapImage
<s:ColorTransform>	flash.geom.ColorTransform
<s:Ellipse>	spark.primitives.Ellipse
<s:GradientEntry>	mx.graphics.GradientEntry
<s:Graphic>	spark.primitives.Graphic
<s:Line>	spark.primitives.Line
<s:LinearGradient>	mx.graphics.LinearGradient
<s:LinearGradientStroke>	mx.graphics.LinearGradientStroke
<s:Path>	spark.primitives.Path
<s:RadialGradient>	mx.graphics.RadialGradient
<s:RadialGradientStroke>	mx.graphics.RadialGradientStroke
<s:Rect>	spark.primitives.Rect
<s:SolidColor>	mx.graphics.SolidColor
<s:SolidColorStroke>	mx.graphics.SolidColorStroke
<s:RichText>	spark.primitives.RichText
<s:Transform>	mx.geom.Transform
<s:BevelFilter>	spark.filters.BevelFilter
<s:BlurFilter>	spark.filters.BlurFilter
<s:ColorMatrixFilter>	spark.filters.ColorMatrixFilter
<s:DropShadowFilter>	spark.filters.DropShadowFilter
<s:GlowFilter>	spark.filters.GlowFilter
<s:GradientGlowFilter>	spark.filters.GradientGlowFilter
<s:GradientBevelFilter>	spark.filters.GradientBevelFilter
<s:ShaderFilter>	spark.filters.ShaderFilter

Vous pouvez écrire le code des dessins MXML en utilisant un éditeur de code ou vous pouvez utiliser un outil comme Flash Catalyst pour générer le code. Généralement, vous utilisez les dessins MXML pour créer des skins pour les composants Flex 4.

Voici un exemple de dessins MXML qui créé cette forme:



```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx">
  <s:Group x="50" y="50">
    <s:Rect height="75" radiusX="5.62472" radiusY="5.99958" width="75"
x="0.5" y="0.5">
      <s:fill>
        <s:SolidColor color="#494848"/>
      </s:fill>
      <s:stroke>
        <s:SolidColorStroke caps="none" joints="miter"
miterLimit="10" weight="1"/>
      </s:stroke>
      <s:filters>
        <s:GlowFilter alpha="0.56" blurX="7.5" blurY="7.5"
color="#777777" inner="true" quality="2" strength="2"/>
      </s:filters>
    </s:Rect>
    <s:Group x="22" y="28">
      <s:Line x="0" xFrom="37" y="14"/>
      <s:Path data="M 36.785 11.654 L 23.489 11.654 L 22.785 11.654
L 22.785 0 L 0 13.154 L 22.785 26.309 L 22.785 14.654 L 23.489 14.654 L 36.785
14.654 L 36.785 11.654 Z" winding="nonZero" x="0.215" y="0.346">
        <s:fill>
          <s:SolidColor color="#FFFFFF"/>
        </s:fill>
      </s:Path>
    </s:Group>
  </s:Group>
</s:Application>
```

## FXG

FXG est un format graphique XML d'échange basé sur la plate-forme Flash. Les autres outils qui connaissent FXG sont Adobe Illustrator et Adobe Photoshop. Vous pouvez créer des éléments graphiques avec ces outils et les exporter sous FXG et les utiliser dans Flex.

Vous ne pouvez pas utiliser le langage FXG à l'intérieur de documents ou composants MXML; à la place, vous définissez des fichiers fxg séparés que vous référencez en MXML. La balise racine d'un document FXG est toujours <Graphic> et il ne peut y avoir qu'un seul noeud <Graphic> dans un document FXG. Par exemple, vous pouvez définir un fichier AlphaMaskComponent.fxg:

```

<?xml version="1.0" encoding="utf-8"?>
<!-- /comps/AlphaMaskComp.fyg -->
<Graphic xmlns="http://ns.adobe.com/fyg/2008" version="2">
  <Ellipse width="400" height="200" maskType="alpha">
    <mask>
      <Group>
        <Rect width="100" height="100">
          <fill>
            <SolidColor alpha="0.1"/>
            10
          </fill>
        </Rect>
      </Group>
    </mask>
    <fill>
      <SolidColor color="#FF00FF"/>
    </fill>
  </Ellipse>
</Graphic>

```

Et l'utiliser en MXML comme ceci:

```

<?xml version="1.0" encoding="utf-8"?>
<s:Application
  xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/mx"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:comps="comps.*">
  <comps:AlphaMaskComponent/>
</s:Application>

```

Beaucoup d'éléments FXG ont des équivalents en dessins MXML bien que des éléments FXG ne supportent qu'un sous-ensemble des attributs pris en charge par des balises de dessins MXML.

Quelles sont les différences et similitudes entre FXG et les dessins MXML ?

- Les deux sont utilisés pour créer des dessins dans les applications Flex.
- FXG génère des dessins compilés (le modèle de rendu FXG suit de très près le modèle de rendu Flash Player 10, et le compilateur optimise FXG directement dans les balises SWF compris nativement par le lecteur Flash). Les dessins MXML sont des graphiques rendus à l'exécution (le compilateur Flex effectue une transformation en ActionScript qui est transformé ensuite en code binaire ActionScript). Ainsi FXG est hautement optimisé par rapport à un même dessin créer en utilisant des dessins MXML.
- Namespaces: FXG utilise son propre namespace ("http://ns.adobe.com/fyg/2008"), quand les dessins MXML utilisent le namespace du document (généralement le namespace Spark).
- FXG est hautement optimisé lors de la compilation, mais comme résultat, il ne peut pas être changé lors de l'exécution. Ainsi si vous voulez l'utiliser pour personnaliser un bouton, vous ne serez pas en mesure d'ajouter des couleurs différentes pour le fond de chaque Etat (over, disabled, up et down). A l'invers de la liaison de données, la gestion des événements est disponible dans FXG.

Vous pouvez lire plus d'informations à cette adresse:

[http://help.adobe.com/en\\_US/flex/using/WS145DAB0B-A958-423f-8A01-12B679BA0CC7.html](http://help.adobe.com/en_US/flex/using/WS145DAB0B-A958-423f-8A01-12B679BA0CC7.html)

## CSS

Flex 4 améliore énormément son support du CSS pour mettre en forme une application Flex. Tout d'abord, vous devez spécifier l'espace de noms de la bibliothèque Flex que vous voulez personnaliser en utilisant CSS. Par exemple, voici un fichier CSS qui définit les styles pour un contrôle Halo Button et un contrôle Spark Button:

```
/* CSS file */
@namespace s "library://ns.adobe.com/flex/spark";
@namespace mx "library://ns.adobe.com/flex/mx";
s|Button {
    color:#ffffff;
}
mx|Button {
    color:#ffffff;
}
```

Vous pouvez utiliser des classes multiples pour le même composant. Supposons que vous ayez défini deux classes CSS appelées *buttonOk* et *buttonGreen*, vous pouvez les appliquer toutes les deux de cette façon:

```
<fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    @namespace mx "library://ns.adobe.com/flex/mx";
    .buttonOK {
        font-size: 12;
    }
    .buttonGreen {
        color: #00ff00;
    }
</fx:Style>
<s:Button styleName="buttonOk buttonGree"/>
```

Vous pouvez définir et appliquer des styles en utilisant des sélecteurs d'ID:

```
<fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    @namespace mx "library://ns.adobe.com/flex/mx";
    #myButton {
        color:#ffffff;
    }
</fx:Style>
<s:Button id="myButton"/>
```

Vous pouvez utiliser des sélecteurs de descendance. Par exemple, vous pouvez appliquer un style uniquement sur les boutons d'un conteneur Spark Group:

```
<fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    @namespace mx "library://ns.adobe.com/flex/mx";
    s|Panel s|Button {
        color:#ffffff;
    }
</fx:Style>
<s:Panel>
    <s:Button label="White label"/>
</s:Panel>
<s:Button label="Black label"/>
```

Vous pouvez utiliser des pseudo sélecteurs:

```
<fx:Style>
  @namespace s "library://ns.adobe.com/flex/spark";
  @namespace mx "library://ns.adobe.com/flex/mx";
  s|Button:over {
    color: #000000;
  }
  s|Button:up {
    color: #ffffff;
  }
</fx:Style>
<s:Button label="My Button"/>
```

Vous pouvez appliquer une feuille de style à l'exécution en compilant le fichier CSS dans un fichier SWF et charger le fichier SWF en utilisant la méthode `loadStyleDeclaration()` de la classe `StyleManager`. Voici les étapes de base pour y parvenir:

- Créer un fichier CSS; par exemple `myStyle.css`
- Dans Flash Builder, choisir le fichier dans l'explorateur de projets et faites un clic droit dessus. Depuis le menu contextuel, sélectionner *Compile CSS to SWF*. Vous devriez trouver un fichier `myStyle.swf` à l'intérieur du dossier `bin-debug`.
- Lorsque vous voulez charger cette feuilles de styles à l'exécution, vous effectuez un appel comme celui-ci:

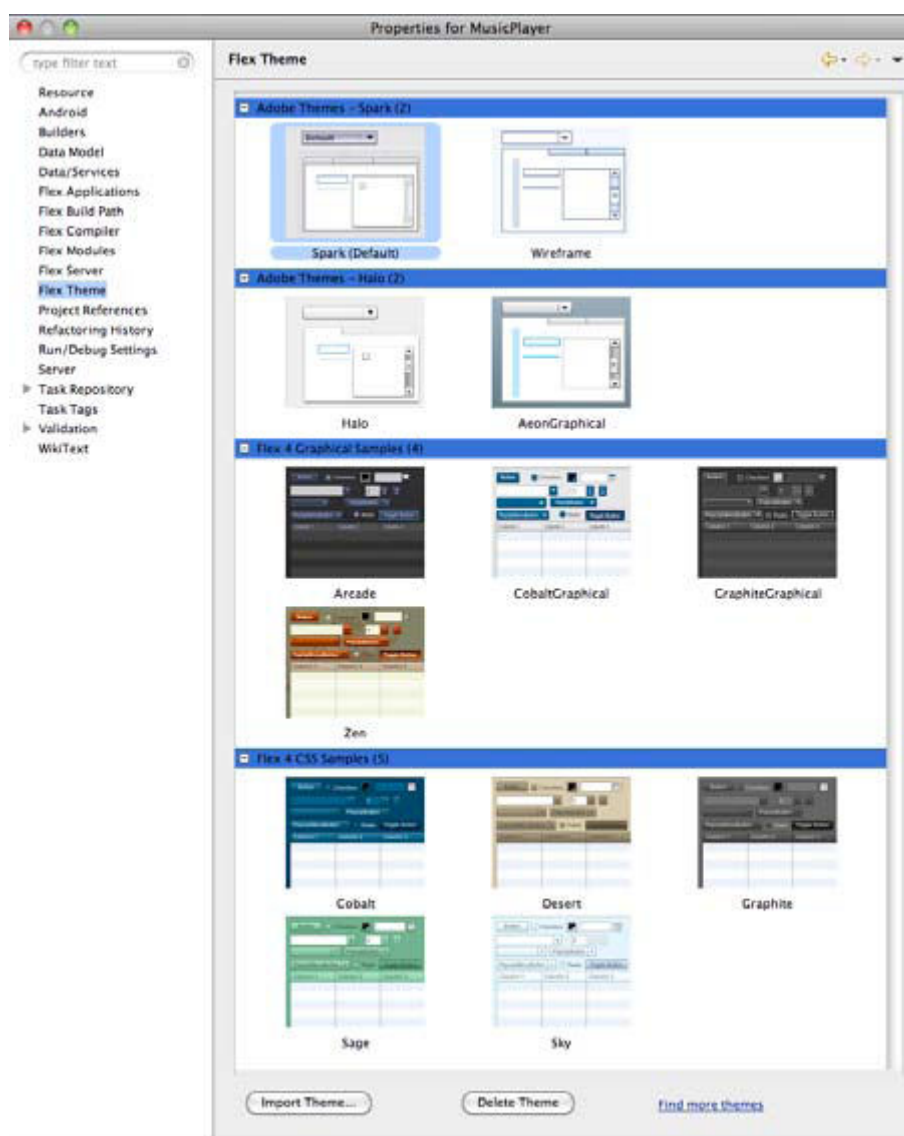
```
styleManager.loadStyleDeclarations("myStyle.swf");
```

Le premier paramètre est le chemin absolu ou relatif vers le fichier SWF et le second paramètre spécifie si la feuille de styles est appliquée immédiatement après son chargement (c'est le comportement par défaut) ou non.

## Thèmes Flex

Un thème définit l'apparence d'une application. Il peut définir seulement quelques styles comme la famille de police, la taille de caractères ou alors il peut redéfinir tous les composants Flex d'interface utilisateur, en utilisant un mélange de CSS, images et skins.

Le thème par défaut pour une application Flex 4 est appelé Spark. Ainsi tous les composants Flex d'interface utilisateur ont par défaut ce thème, y compris les composants Halo (Flex 3 composants). Vous pouvez changer le thème en utilisant l'argument de thème du compilateur. Si vous utilisez Flash Builder 4, alors vous avez une page de propriétés qui facilite l'application d'un thème à une application, ou l'ajout de nouveaux thèmes.



Flash Builder offre un grand choix de thèmes. Si vous n'êtes pas satisfaits d'eux, vous pouvez créer vos propres thèmes. Vous pouvez appliquer de multiples thèmes à un même projet. Cependant, si ces thèmes définissent l'apparence pour le même élément, alors comme avec les CSS, c'est le dernier thème appliqué qui prend le dessus.

Si vous voulez avoir les composants Halo avec l'apparence Flex 3, vous devez utiliser le thème Halo. Veuillez noter que les composants Spark utilisent la même skin que celle définie par le thème Spark.

Si vous voulez voir le code permettant la création d'un thème, aller dans le dossier *Flex 4 SDK/frameworks/themes/*.

## Création d'un thème Flex

En général, les thèmes sont sous la forme d'un fichier SWC. A l'intérieur, vous avez un fichier CSS et des ressources de mise en forme. Les ressources de mise en forme peuvent être des images (JPEG, GIF, PNG) ou des ressources de programmation compilées dans un fichier SWF.

Une autre façon de créer un thème est d'utiliser uniquement un fichier CSS et de l'utiliser sans le compiler dans un fichier SWF. Ou vous pouvez avoir un fichier CSS et un fichier SWF (c'est de cette façon qu'est créé le thème AeonGraphical).

Supposons que vous avez créé ce fichier CSS (SpecialButtonSkin et SpecialCheckBoxSkin sont deux classes de skin de programmation que vous avez créés en MXML; ce sont leurs noms de classe):

```
//mycss.css
@namespace s "library://ns.adobe.com/flex/spark";
s|Button {
    skinClass: ClassReference("SpecialButtonSkin");
}
s|CheckBox {
    skinClass: ClassReference("SpecialCheckBoxSkin");
}
s|panel {
    color:#ff0000;
}
```

Ensuite, vous pouvez utiliser le compilateur de composants en ligne de commande (compc) avec les options `include-file` et `include-classes` pour inclure les ressources images et les skins. Vous utilisez l'option `o` pour spécifier le fichier SWC de sortie. Voici un exemple:

```
compc -source-path c:/projects/flex/mytheme
      -include-file mycss.css c:/projects/flex/mytheme/mycss.css
      -include-classes SpecialButtonSkin SpecialCheckBoxSkin
      -o c:/projects/flex/MyTheme.swc
```

Au lieu de passer tous ces différents arguments dans la ligne de commande, vous pouvez créer un fichier de configuration Flex et le passer comme argument à l'option `load-config` de la commande `compc`.

```
compc -load-config myconfig.xml
```

Voici un exemple d'un fichier de configuration Flex:

```
<?xml version="1.0"?>
<flex-config>
<output>MyTheme.swc</output>
<include-file>
    <name>mycss.css</name>
```

```
<path>c:/projects/flex/mytheme/mycss.css</path>
</include-file>
<include-classes>
<class> SpecialButtonSkin </class>
<class> SpecialCheckBoxSkin</class>
</include-classes>
</flex-config>
```

Une autre façon de créer un thème est d'utiliser un projet Flex Library de Flash Builder pour créer le fichier de thème SWC.

Vous pouvez en lire plus à cette adresse:

[http://help.adobe.com/en\\_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf69084-7f85.html#WS2db454920e96a9e51e63e3d11c0bf69084-7e65](http://help.adobe.com/en_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf69084-7f85.html#WS2db454920e96a9e51e63e3d11c0bf69084-7e65)

## Appliquer un thème

Vous pouvez appliquer un thème en utilisant l'option theme et le compilateur MXML:

```
mxmlc -theme="C:\Program
Files\Adobe\Flex\sdk\4.0.0\frameworks\themes\Halo\halo.swc" MyApp.mxml
```

Sinon vous pouvez utiliser les propriétés Flash Builder du projet que vous souhaitez personnaliser et sélectionner le thème (s'il ne l'est pas déjà, vous pouvez aussi importer un thème depuis cette page).

Une dernière remarque, c'est que vous ne pouvez pas appliquer un thème en ajoutant simplement le fichier SWC dans le dossier libs. Si vous voulez appliquer des thèmes à l'exécution, vous devez lire le chapitre CSS (typiquement vous utilisez un fichier SWF à l'intérieur du fichier SWC et appelez la méthode *loadStyleDeclarations* de la classe *StyleManager*).

## Composants Spark

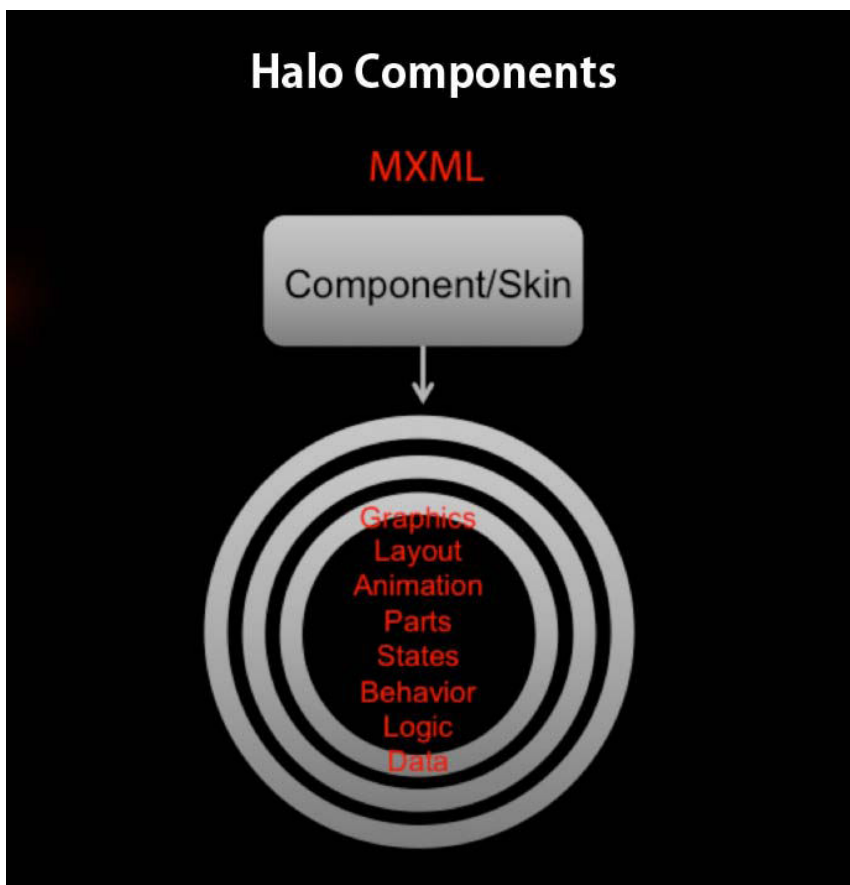
La plupart des fonctionnalités présentées jusqu'ici peuvent être considérées comme des évolutions: petites améliorations ou ajouts. Les composants Spark représentent eux une fonctionnalité révolutionnaire. La nouvelle architecture des composants (nom de code Spark), introduit par Flex 4, est clairement une rupture avec le précédent modèle de composant de Flex 3 et 2 (nom de code Halo).

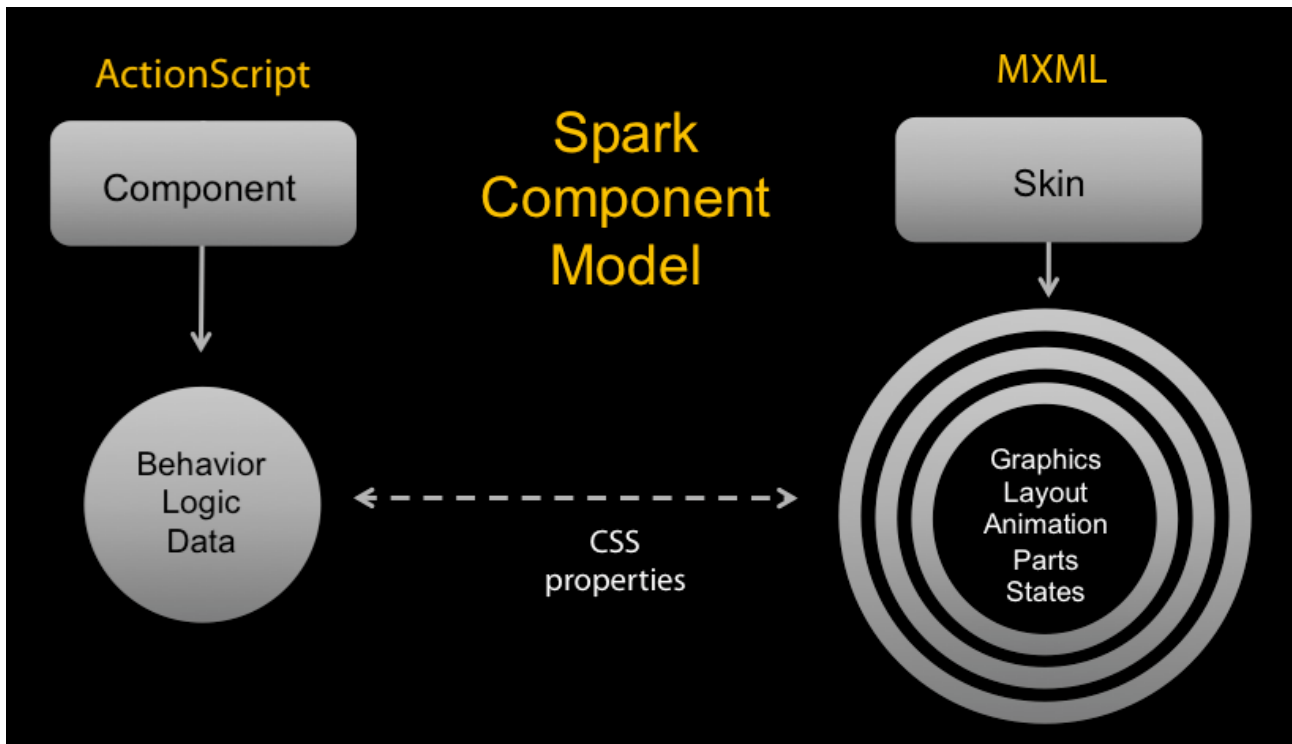
En un mot, les composants Spark introduisent une séparation claire entre la logique métier et les données d'un composant et son aspect visuel (skin). Un composant Spark a au moins deux classes.

Une classe (généralement écrite en ActionScript) donne le nom de balise MXML du composant (par exemple, Button) et encapsule le comportement de base du composant: définition des événements diffusés par le composant, des données représentant le composant (pour un bouton cela pourrait être le text du bouton), et gestion des sous-composants ou des différents états (pour un bouton cela être: haut, bas, au-dessus et désactivé).

Couplé à cette classe il ya une classe de skin qui gère tout ce qui est lié à la façon dont le composant est rendu à l'écran, y compris la mise en page, les animations, le changement d'apparence dans les différents états, les transitions et la représentation des données. Généralement vous créez une classe d'habillage en utilisant les graphiques MXML et/ou FXG.

Du point de vue des outils, vous pouvez utiliser un éditeur de texte pour créer la classe de Skin ou vous pouvez utiliser le nouveau flux de travail introduit avec Flash Catalyst ou Adobe Illustrator/Photoshop. Ce flux de travail permet à un concepteur ou à un développeur de créer visuellement les skins à l'aide de ces outils.





Flex 4 nécessite une version minimum de Flash Player 10 afin de faire fonctionner une application Flex (dans Flash Player 10 ont été introduites le nouveau moteur de rendu de texte FTE et Pixel Bender, fonctionnalités utilisées par Flex 4).

Voici une liste de composants Spark:

Application	RadioButton
BorderContainer	RadioButtonGroup
Button	RichEditableText
ButtonBar	RichText
CheckBox	Scroller
DataGroup	SkinnableContainer
DataRenderer	SkinnableDataContainer
DropDownList	Spinner
Group	TextArea
HGroup	TextInput
HScrollBar	ToggleButton
HSlider	VGroup
Label	VideoPlayer
List	VScrollBar
NavigatorContent	VSlider
NumericStepper	Window
Panel	WindowedApplication

Si vous regardez la liste ci-dessus, vous remarquerez qu'il n'y a pas un composant Spark pour chaque composant Halo. C'est quelque chose sur lequel Adobe travaille afin de l'intégrer dans les prochaines versions du SDK Flex. Dans le même temps, il n'y aura pas un composant Spark équivalent pour chaque composant Halo. Toujours dans Flex 4, vous disposez de tous les composants Flex 3 et vous pouvez mélanger, si vous le souhaitez, ces deux ensembles de composants dans le même projet.

Les composants Spark sont des sous-classes de *UIComponent* (comme les composants Halo). Cependant, une série de nouvelles classes qui héritent de *UIComponent* ont été introduites et elles forment les classes de base pour tous les composants Spark.

Si vous voulez créer un composant Spark qui doit être personnalisable, alors vous devez créer une sous-classe de *spark.components.supportClasses.SkinnableComponent*.

Si vous voulez créer un conteneur Spark qui doit être personnalisable, vous devez étendre la classe *spark.components.SkinnableContainer*. Il existe des conteneurs Spark qui ne sont pas personnalisables, comme par exemple le conteneur *Group*.

Deux autres conteneurs sont importants: *DataGroup* et *SkinnableDataContainer*. Ces conteneurs peuvent être utilisés pour afficher des données. Par exemple, si vous voulez créer une liste, vous pouvez utiliser un conteneur *SkinnableDataContainer*. Si vous ne voulez pas utiliser de skin, vous pouvez utiliser un conteneur *DataGroup*.

Tous les conteneurs Spark supportent des enfants qui implémentent l'interface *IVisualElement*. Les composants Spark et Halo et les classes *spark.primitives.supportClasses.GraphicElement* implémentent cette interface. Ainsi, vous pouvez ajouter à un conteneur Spark: des composants d'interface (Spark ou Halo), des dessins MXML ou FXG.

Si vous voulez créer une skin pour un conteneur Spark, typiquement vous créez un fichier MXML contenant un *Group* dont l'id a pour valeur *contentGroup* et quelques balises de dessins MXML ou FXG pour créer l'apparence.

Enfin, si vous voulez créer une skin pour un composant Spark, vous devez étendre la classe *spark.components.supportClasses.Skin*.

## **Le contrat de personnalisation – Personnalisation des composants Spark**

La séparation dans deux classes différentes de la logique métier et de l'apparence d'un composant est une bonne chose, mais s'il n'y a pas d'interactions entre ces classes, nous perdons une grande partie des interfaces riches. Imaginez une belle skin pour un bouton qui ne réagit pas aux événements de survol ou de clic.

Afin de permettre l'interactivité entre le composant et sa skin, il existe un contrat entre ces deux entités. Le contrat de personnalisation a ces trois composantes: données, parties, et états.

Prenons un simple composant Spark, comme un bouton et analysons ces composantes.

## Button component

## Skin component

### Data:

```
[Bindable]
```

```
public function set label
```

```
<s:Label text="{hostComponent.label}"/>
```

### Parts:

```
[SkinPart(required="false")]
```

```
public var labelDisplay:TextBase;
```

```
<s:Label id="labelDisplay"/>
```

### States:

```
[SkinState("up")]
```

```
[SkinState("over")]
```

```
[SkinState("down")]
```

```
[SkinState("disabled")]
```

```
<s:State name="up"/>
```

```
<s:State name="over"/>
```

```
<s:State name="down"/>
```

```
<s:State name="disabled"/>
```

## Les données

La classe de composant contient les données. Pour un bouton, ce pourrait être la légende (texte) du bouton:

```
[Bindable]
public function set label
```

La classe de skin définit un composant *Label* pour afficher les données sur l'écran. Il peut lire les données de la classe du composant en utilisant la variable *hostComponent*:

```
<s:Label text="{hostComponent.label}"/>
```

## Les parties (parts)

La classe de composant définit les parties de la skin et si elles sont obligatoires ou optionnelles:

```
[SkinPart(required="false")]
public var labelDisplay:TextBase;
```

La classe de skin implémente ses parties (ou elle peut choisir de les ignorer si elle sont optionnelles):

```
<s:Label id="labelDisplay"/>
```

## Les états

La classe de composant définit les états qui sont supportés:

```
[SkinState("up")]
[SkinState("over")]
```

```
[SkinState("down")]
[SkinState("disabled")]
```

La classe de skin peut choisir ou non de réagir aux changements d'états. Mais elle doit déclarer les états définis par le composant hôte:

```
<s:State name="up"/>
<s:State name="over"/>
<s:State name="down"/>
<s:State name="disabled"/>
```

Relations entre le composant et sa skin: vous pouvez définir une relation entre une classe de skin et le composant qui sera personnalisé avec elle. Vous pouvez ajouter à la classe de skin cette méta-données:

```
<fx:Metadata>
    [HostComponent("spark.components.Button")]
</fx:Metadata>
```

## Création de composants et de skins

Vous avez deux méthodes pour créer un composant Spark et une skin:

- Partir de rien; vous étendez les classes *SkinnableComponent* et *Skin*.
- Etendre un composant Spark existant ou une skin existante.

Voici le code pour un composant personnalisé qui peut être utilisé pour afficher le statut d'une connexion réseau.

Le code pour le composant:

```
package org.corlan {
    import spark.components.supportClasses.SkinnableComponent;
    import spark.components.supportClasses.TextBase;
    //the states defined by the skin
    [SkinState("connected")]
    [SkinState("disconnected")]
    [SkinState("disabled")]
    public class ConnectionStatus extends SkinnableComponent {
        private var _content:String;
        private var _enabled:Boolean = true;
        private var _connected:Boolean = false;
        [SkinPart(required="true")]
        public var labelDisplay:TextBase;
        public function ConnectionStatus() {
            super();
        }
        public function set label(val:String):void {
            _content = val;
        }
        public function get label():String {
            return (_content != null) ? _content.toString() : "";
        }
        public function set connected(val:Boolean):void {
            _connected = val;
            invalidateState();
        }
    }
}
```

```

public function get connected():Boolean {
    return _connected;
}
override public function set enabled(val:Boolean):void {
    enabled = val;
    super.enabled = val;
}
private function set content(val:String):void {
    _content = val;
    21
    // Push to the optional labelDisplay skin part
    if (labelDisplay)
        labelDisplay.text = label;
}
private function get content():String {
    return _content;
}
override protected function partAdded(partName:String,
instance:Object):void {
    super.partAdded(partName, instance);
    if (instance == labelDisplay) {
        // Push down to the part only if the label was
explicitly set
        if (_content)
            labelDisplay.text = label;
    }
}
override protected function getCurrentSkinState():String {
    if (!enabled)
        return "disabled";
    if (_connected)
        return "connected";
    else
        return "disconnected";
}
private function invalidateState():void {
    //invalidateProperties();
    invalidateSkinState();
}
}
}

```

Le code pour la classe de skin:

```

<?xml version="1.0" encoding="utf-8"?>
<s:SparkSkin xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
alpha.disabled="0.5">
    <fx:Metadata>
        [HostComponent("org.corlan.ConnectionStatus")]
    </fx:Metadata>
    <s:states>
        <s:State name="disconnected"/>
        <s:State name="connected" />
        <s:State name="disabled" />
    </s:states>
    <s:Group>
        <s:layout>
            <s:HorizontalLayout/>
        </s:layout>
        <s:Label id="labelDisplay"
            textAlign="center"
            verticalAlign="middle"

```

```

        maxDisplayedLines="1"
        horizontalCenter="0" verticalCenter="1" color="#ff0000"
color.connected="#00ff00"/>
    </s:Group>
</s:SparkSkin>

```

Et vous pouvez utiliser ce composant comme ceci:

```

private function changeConnStatus():void {
    connStatus.connected = ! connStatus.connected;
}
<corlan:ConnectionStatus id="connStatus"
    label="Connection Status"
    skinClass="org.corlan.ConnectionStatusSkin"/>

```

## Cycle de vie de composants et de skins

Comment la classe de skin communique-t-elle avec la classe du composant ? La classe du composant doit implémenter la méthode `getCurrentSkinState()`. Cette méthode retourne l'état que la skin doit afficher. Quand vous voulez pousser les données dans la skin, vous appelez la méthode `invalidateSkinState()` de la classe du composant. À son tour, la skin fera appel à la méthode `getCurrentSkinState()` et va changer son état en conséquence.

Qu'en est-il de l'injection des données du composant vers la skin (par exemple la chaîne de l'étiquette d'un bouton) ? Il ya deux façons d'atteindre cet objectif. La skin peut extraire les données en utilisant la variable `hostComponent`. Ou l'objet composant peut pousser les données dans la skin, à l'aide des parties de skin (`skinPart`). Si vous souhaitez utiliser cette dernière méthode, vous devez surcharger la méthode `partAdded()` (vous pouvez voir cette approche utilisée dans l'exemple précédent).

<i>Component</i>	<i>Skin</i>
<pre> Constructor(), ... commitProperties() loadSkin() </pre>	<pre> Constructor() </pre>
<pre> Skin.hostComponent = this addChild(skin) findSkinParts() partAdded() getCurrentSkinState() measure()/updateDisplayList()/ updateComplete </pre>	<pre> initialize() currentState = ... commitProperties()/measure()/ updateDisplayList()/updateComplete </pre>
<pre> createDynamicPartInstance(), removeDynamicPartInstance(), getDynamicPartAt(), numDynamicParts(). </pre>	

## Ressources

Ajouter un nouvel état à une skin d'un composant qui étend un composant existant.

<http://saturnboy.com/2010/06/drawer-component-flex-4/>

Introduction à la personnalisation (skinning)

[http://www.adobe.com/devnet/flex/articles/flex4\\_skinning.html](http://www.adobe.com/devnet/flex/articles/flex4_skinning.html)

## Le nouveau moteur de mise en page

Dans Flex 3, la mise en page était fortement couplée avec le conteneur à l'intérieur des contrôles. Chaque type de conteneur exposait un comportement de mise en page. Les composants partageant les mêmes fonctionnalités étaient dupliqués dans le but d'atteindre des comportements de mise en page différents. Ainsi, il était difficile de créer des composants travaillant avec des collections et présentant leurs enfants de différentes manières.

Dans Flex 4, cette problématique est résolue par la séparation de la mise en page et du composant. Les composants Spark définissent leur mise en page de façon déclarative. La mise en page peut supporter à la fois des composants Spark et Halo en plus des primitives graphiques FXG. La mise en page peut être modifiée à l'exécution.

Il ya quelques fonctionnalités supplémentaires à la disposition des développeurs:

1. La logique de mise en page est définie dans une classe séparée qui hérite de la classe *LayoutBase* (la classe mère);
2. L'interface *ILayoutElement* a été créée pour autoriser les classes de mise en page à gérer le conteneur dans lequel elles sont appliquées;
3. Les conteneurs de données Spark possèdent une mise en page virtuelle héritée de la classe mère *DataGroup*. La virtualisation est destinée à minimiser les ressources et le temps de démarrage pour les conteneurs avec un grand nombre d'éléments. Quand vous utilisez la virtualisation, le calcul des dimensions du conteneur *DataGroup* est approximativement basé sur les dimensions de mise en page du premier élément de la propriété *itemRendered* ou sur les dimensions de l'élément typique spécifié via la propriété *typicalItem*.
4. Pour les conteneurs Spark, l'ordre d'affichage est dissocié de l'ordre des enfants.

Comme une conséquence directe de ces fonctionnalités, les mises en page supportent les transformations 2D et le défilement lissé par pixel. Les transformations post mise en page permettent aux développeurs de spécifier les propriétés de position sans affecter la mise en page. En outre des mises en page personnalisées peuvent être créées très facilement et des transformations 3D peuvent être incorporées dans ces mises en page.

### Les mises en page Spark existantes

Flex est fourni avec plusieurs classes de mise en page qui peuvent être utilisées avec les conteneurs Spark. Les classes de mise en page sont définies dans le package *spark.layout*.

#### BasicLayout

La classe *BasicLayout* utilise le positionnement absolu. L'enfant peut être positionné soit en utilisant des coordonnées, soit en utilisant des contraintes.

Cette mise en page n'est pas supportée par les contrôles Spark de type liste. Il ne faut pas utiliser cette mise en page avec le contrôle Spark List et l'une de ses sous-classes (comme *ButtonBar*, *ComboBox*, *DropDownList* ou *TabBar*). La classe *BasicLayout* ne supporte la mise en page virtuelle.

```

<Group>
  <layout>
    <BasicLayout/>
  </layout>
  <Label text="content" baseline="40"/>
</Group>

```

## HorizontalLayout

La classe `HorizontalLayout` dispose ses enfants les uns à côté des autres, de gauche à droite, avec un espace optionnel entre eux et un espace optionnel autour d'eux.

Durant l'exécution de la méthode de calcul des dimensions (`measure()`), la taille par défaut du conteneur est calculé en cumulant les dimensions préférées des éléments et les espacements.

```

<s:List dataProvider="{objectsList}" itemRenderer="components.CustomerRendered"
width="200">
  <s:layout>
    <s:HorizontalLayout clipAndEnableScrolling="true" />
  </s:layout>
</s:List>

```

## TileLayout

La classe `TileLayout` dispose ses enfants en lignes et en colonnes, dans des cellules de dimensions égales. La classe `TileLayout` utilise un certain nombre de propriétés qui contrôlent l'orientation, le nombre, la taille, l'écart et la justification des colonnes et les lignes ainsi que l'alignement des éléments dans les cellules.

```

<s:List dataProvider="{objectsList}" itemRenderer="components.CustomerRendered"
width="200" height="50">
  <s:layout>
    <s:TileLayout />
  </s:layout>
</s:List>

```

## VerticalLayout

La classe `VerticalLayout` dispose ses enfants les uns sous les autres, de haut en bas, avec un espace optionnel entre eux et un espace optionnel autour d'eux.

Durant l'exécution de la méthode de calcul des dimensions (`measure()`), la taille par défaut du conteneur est calculé en cumulant les dimensions préférées des éléments et les espacements.

```

<s:List dataProvider="{objectsList}" itemRenderer="components.CustomerRendered">
  <s:layout>
    <s:VerticalLayout gap="15" />
  </s:layout>
</s:List>

```

## Création d'une mise en page Spark personnalisée

Afin de créer une mise en page Spark personnalisée, on doit hériter de la classe de base `LayoutBase` et surcharger au moins une méthode - la méthode `updateDisplayList`.

Quand une mise en page est assignée à un conteneur, la propriété `target` de la mise en page reçoit une référence au conteneur. La mise en page personnalisée peut décider sur le positionnement et la taille des éléments à l'intérieur du conteneur. Pour un tel scénario, une attention particulière doit être accordée au type du conteneur. Si le conteneur supporte la virtualisation, la recherche d'un élément dans le conteneur est réalisable grâce à la méthode `getVirtualElementAt` de l'objet cible tandis que pour les conteneurs ne supportant pas la virtualisation, la méthode `getElementAt` doit être utilisée.

Le redimensionnement et positionnement des éléments est assez simple grâce à l'utilisation de l'API `ILayoutElement`.

```
public class Horizontal3DLayout extends LayoutBase
{
    override public function updateDisplayList(width:Number,
height:Number):void
    {
        var numElements:int = target.numElements;
        var targetWidth:Number = target.width;
        var middleHeight:Number = target.height / 2;
        for (var i : int = 0; i < numElements; i++)
        {
            var element:ILayoutElement = target.getVirtualElementAt(i);
            element.setLayoutBoundsSize(NaN, NaN, false);
            var elWidth:Number = NaN;
            var elHeight:Number = NaN;
            if (i < numElements/2)
            {
                element.setLayoutMatrix(new Matrix(1.6, 0, 0, 1.6, 0,
0), true);

                elWidth = element.getLayoutBoundsWidth(true);
                elHeight = element.getLayoutBoundsHeight(false);
                element.setLayoutBoundsPosition(i * elWidth ,
middleHeight + elHeight, true);
            }
            else
            {
                element.setLayoutMatrix(new Matrix(0.9, 0, 0, 0.9, 0,
0), true);

                elWidth = element.getLayoutBoundsWidth(true);
                elHeight = element.getLayoutBoundsHeight(false);
                element.setLayoutBoundsPosition(targetWidth - i *
elWidth , middleHeight - elHeight, true);
            }
        }
    }
}
```

### En savoir plus

La documentation en ligne est un bon endroit pour commencer, spécialement les références dédiées aux moteurs de rendu, skins, `LayoutBase`, `ILayoutElement`, `GroupBase`, `DataGroup` et la virtualisation. En outre, vérifier les sources ci-dessous.

## Ressources

[http://www.adobe.com/devnet/flex/articles/spark\\_layouts.html](http://www.adobe.com/devnet/flex/articles/spark_layouts.html)

<http://insideria.com/2009/05/flex-4-custom-layouts.html>

<http://www.slideshare.net/rjowen/flex-4-overview>

[http://www.adobe.com/devnet/flex/articles/flex3and4\\_differences\\_print.html](http://www.adobe.com/devnet/flex/articles/flex3and4_differences_print.html)

## Le nouveau moteur d'animation

Tous les nouveaux effets Flex 4 sont des sous-classes de la classe Animation, qui est une sous-classe de la classe Effect. Cela permet au framework Flex 4 de proposer une hiérarchie parallèle d'effets qui ne casse pas la compatibilité avec les anciens effets, puisque ceux-ci sont des sous-classes de la classe TweenEffect. Les fonctionnalités exposées par la classe Animation sont similaires à celles de la classe Tween dans Flex 3

Les nouvelles classes pour l'animation sont groupées dans le package spark.effects. L'animation peut-être un changement de position (effectué par l'effet Move), un changement de taille (effectué par l'effet Resize), un changement de visibilité (effectué par l'effet Fade) ou tous autres types d'animations réalisées par des effets ou exécutées directement par le biais de la classe Animation..

Quand vous définissez des effets d'animations, vous créez une instance de la classe Animate, ou d'une sous-classe de la classe Animate. Cela crée une instance de la classe Animation dans la méthode play (). L'instance de la classe Animation accepte des valeurs de début et de fin, une durée et des paramètres optionnels tels que des objets d'accélération ou d'interpolations.

```
<s:Animate id="mover" target="{button}" duration="1000">
    <s:SimpleMotionPath property="x" valueFrom="0" valueTo="100" />
    <s:SimpleMotionPath property="y" valueTo="100" />
    <s:SimpleMotionPath property="width" valueBy="20" />
</s:Animate>
...
<s:Button id="button" click="mover.play()" />
```

La classe Animation est un moteur de temps. Cette classe calcule les propriétés sur la base des propriétés qui lui sont passées, elle n'anime pas les objets par elle-même. Une cible d'animation lui est passée et est responsable de consommer les nouvelles propriétés (et d'animer les objets).

```
<s:Elastic id="elastic" />
<s:Animation id="animation" animationTarget="{new
utilities.PropertyAnimator(rect)}" easer="{elastic}" duration="1200">
    <s:SimpleMotionPath property="x" valueFrom="0" valueTo="200" />
    <s:SimpleMotionPath property="y" valueFrom="0" valueTo="200" />
</s:Animation>
```

### Construire des effets

Flex 4 comprend plusieurs sous-classes de la classe Animate permettant la manipulation des effets les plus utilisés. Ces effets sont responsables de fournir les propriétés à la classe Animate, puis d'exécuter toutes les fonctionnalités destinées à animer l'objet ciblé. La convention de nommage de ces effets est assez suggestive, les noms décrivent le type d'animation réalisé par chaque effet.

- a. L'effet Resize – change la largeur, la hauteur ou les deux dimensions d'un composant. \\Quand les valeurs de départ du redimensionnement ne sont pas fournies, les valeurs actuelles de l'objet sont utilisées. Spécifier 2 propriétés parmi *from*, *to* et *by* demande au moteur de calculer la troisième propriété.

```
<s:Resize id="resizer" target="{rect}" widthTo="200" heightTo="200" />
...
<s:Button id="resizeBtn" label="Resize effect" click="resizer.play()" />
```

- b. Les effets de transformation (Move, Rotate, Scale) – ces effets ne fonctionnent que dans des sous-classes de `UIComponent` et `GraphicElement`. Les effets de transformation opèrent par rapport au centre de transformation de la cible. Par défaut, c'est le coin haut gauche de la cible. Pour jouer la transformation autour du point central de la cible, il faut définir la propriété `autoCenterTransform` sur la valeur `true`.

```
<s:Move id="mover" target="{rect}" xTo="200" yTo="200" />
<s:Rotate id="rotator" target="{rect}" angleTo="120"
autoCenterTransform="true" />
<s:Scale id="scaler" target="{rect}" scaleXBy="2" scaleYBy="2"
autoCenterTransform="true" />
```

...

```
<s:Button id="moveBtn" label="Move effect" click="mover.play()" />
<s:Button id="rotateBtn" label="Rotate effect" click="rotator.play()" />
<s:Button id="scaleBtn" label="Scale effect" click="scaler.play()" />
```

- c. L'effet `Fade` – cet effet anime la propriété `alpha` d'un composant.

```
<s:Fade id="fader" target="{rect}" alphaTo="0.5" />
```

...

```
<s:Button id="fadeBtn" label="Fade effect" click="fader.play()" />
```

- d. L'effet `AnimateColor` – anime un changement dans la propriété de couleur au fil du temps, avec une interpolation entre les données des valeurs des propriétés `colorFrom` et `colorTo` sur une base par canal

```
<s:AnimateColor id="colorEffect" target="{myGradient}"
colorFrom="0x0000FF" colorTo="0xFF0000" repeatCount="2"
repeatBehavior="reverse" />
```

...

```
<s:Button id="clrBtn" label="Color effect" click="colorEffect.play()" />
```

- e. L'animation `Keyframe` – elle est réalisée avec l'aide des classes `MotionPath` and `Keyframe`. Les objets `Keyframe`, spécifiés en tant qu'enfants d'une instance `MotionPath`, sont des paires clés/valeurs dont une propriété va prendre la valeur à un instant précis de l'animation.

```
<s:Animate id="colorKeyframes" target="{myGradient}">
  <s:MotionPath property="alpha">
    <s:Keyframe time="500" value="1.0" />
    <s:Keyframe time="500" value="0.3" />
    <s:Keyframe time="1100" value="0.0" />
    <s:Keyframe time="300" value="0.3" />
    <s:Keyframe time="300" value="0.5" />
    <s:Keyframe time="300" value="1.0" />
  </s:MotionPath>
  <s:MotionPath property="color">
    <s:Keyframe time="1000" value="0x00FF00" />
    <s:Keyframe time="2000" value="0xCCCCCC" />
  </s:MotionPath>
</s:Animate>
```

...

```
<s:Button id="clrKeyframesBtn" label="Keyframe animate effect"
click="colorKeyframes.play()" />
```

## Effets avancés

- a. Les effets 3D – Ces effets ajoutent le support de l'axe `z` dans l'exécution d'une animation. Augmenter la valeur `z` fera s'éloigner l'objet du spectateur, la diminuer aura l'effet inverse, l'objet se déplacera vers le spectateur. Comme pour les effets de transformation 2D, l'utilisation des effets de transformation 3D peut modifier la mise en page du conteneur parent de l'objet animé. Pour modifier ce comportement, il faut définir la propriété `applyChangePostLayout` de

l'effet sur la valeur false.

```
<s:Move3D id="moverBis" duration="600" xBy="-200" zBy="-200"
autoCenterTransform="true" repeatCount="2" repeatBehavior="reverse"
target="{rect}" />
<s:Rotate3D id="rotatorBis" target="{rect}" angleXFrom="0" angleXTo="180"
angleYFrom="0" angleYTo="180" duration="1000" autoCenterTransform="true"
/>
<s:Scale3D id="scalerBis" target="{rect}" scaleXBy="-0.25" />
...
<s:Button id="moveBtnBis" label="3D Move effect" click="moverBis.play()"
/>
<s:Button id="rotateBtnBis" label="3D Rotate effect"
click="rotatorBis.play()" />
<s:Button id="scaleBtnBis" label="3D Scale effect"
click="scalerBis.play()" />
```

- b. Les effets Pixel Shader - Ces effets sont utilisés pour appliquer une animation à un objet cible qui a une représentation bitmap avant et après l'animation. Travailler avec des objets Image semble évident, mais les effets de Shader étendent leurs capacités à animer un composant Flex. Afin d'atteindre cet objectif, ils capturent une image bitmap du composant, avant et après l'animation et appliquent celle-ci entre ces deux images.

```
[Embed(source="assets/Blue hills.jpg")]
private const ImgA:Class;
[Embed(source="assets/Sunset.jpg")]
private const ImgB:Class;
...
<s:CrossFade id="crossFade" bitmapFrom="{new ImgA().bitmapData}"
bitmapTo="{new ImgB().bitmapData}" target="{faded}" duration="2000"
repeatCount="2" repeatBehavior="reverse" />
<s:Wipe id="wipe" bitmapFrom="{new ImgA().bitmapData}" bitmapTo="{new
ImgB().bitmapData}" target="{faded}" duration="2000" repeatCount="2"
repeatBehavior="reverse" />
...
<s:Button id="crossFadeBtn" label="Cross fade effect"
click="crossFade.play()" />
<s:Button id="wipeBtn" label="Wipe effect" click="wipe.play()" />
```

- c. Les effets AnimateFilter - Les effets AnimateFilter diffèrent des autres effets parce qu'ils permettent d'animer les propriétés d'un filtre appliqué à un objet cible plutôt que les propriétés de l'objet cible. Les filtres sont inclus dans le même paquet; les filtres courants sont DropShadowFilter, GlowFilter, BlurFilter et ShaderFilter.

```
<s:AnimateFilter id="animF" target="{shadowRect}" bitmapFilter="{new
spark.filters.DropShadowFilter()}">
  <s:SimpleMotionPath property="color" valueFrom="0"
valueTo="0x0000FF" />
  <s:SimpleMotionPath property="distance" valueFrom="0" valueTo="20"
/>
  <s:SimpleMotionPath property="angle" valueFrom="270" valueTo="360"
/>
</s:AnimateFilter>
...
<s:Button id="animFilterBtn" label="AnimateFilter effect"
click="animF.play()" />
```

- d. Liaisons d'effets (Parallel vs Sequence) - Les effets peuvent être liés pour être joués soit en parallèle soit en séquence. Les nouvelles implémentations Spark ont été fournies pour les effets composites de type Parallel ou Sequence. Les effets sont utilisés pour faire exactement

ce que leur nom laisse entendre, soit exécuter les effets enfants en parallèle ou soit les exécuter en une séquence d'effets, les uns après les autres. Sequence peut également permettre d'exécuter des actions d'effets au cours d'une transition.

```
<s:Sequence id="seq">
  <s:Move target="{rect}" xBy="200" />
  <s:Rotate target="{rect}" angleBy="360" autoCenterTransform="true"
/>
  <s:Move target="{rect}" xBy="-200" />
</s:Sequence>
<s:Parallel id="paralleleffect">
  <s:Rotate target="{rect}" angleTo="360" autoCenterTransform="true"
/>
  <s:AnimateColor target="{myGradient}" colorFrom="0x0000FF"
colorTo="0xFF0000" repeatCount="2" repeatBehavior="reverse" />
</s:Parallel>
...
<s:Button id="paralleleffectBtn" label="Paralleleffect"
click="paralleleffect.play()" />
<s:Button id="seqBtn" label="Sequence effect" click="seq.play()" />
```

Les effets Spark supportent tous les événements principaux tels que `effectStart`, `effectStop` et `effectEnd`. Ils supportent aussi `effectRepeat` diffusé lorsque l'effet commence une nouvelle répétition et `effectUpdate`, diffusé chaque fois que l'effet met à jour l'objet cible.

On peut changer l'accélération d'une animation en utilisant une classe d'accélération avec l'effet. Flex fournit des classes d'accélération dans le package `spark.effects.easing`. Certains d'entre eux sont Bounce, Elastic, Linear, Power et Sine.

References:

[http://help.adobe.com/en\\_US/flex/using/](http://help.adobe.com/en_US/flex/using/)  
<http://graphics-geek.blogspot.com/2010/02/video-keyframe-animations-in-flex-4.html>  
<http://insideria.com/2010/02/effects-and-animation-hello-fl.html>  
[http://www.adobe.com/devnet/flex/articles/flex4\\_effects\\_pt1\\_print.html](http://www.adobe.com/devnet/flex/articles/flex4_effects_pt1_print.html)  
[http://www.adobe.com/devnet/flex/articles/flex4\\_effects\\_pt2\\_print.html](http://www.adobe.com/devnet/flex/articles/flex4_effects_pt2_print.html)  
<http://blog.flexexamples.com/2010/04/22/creating-a-simple-property-animation-in-flex-4/>  
<http://graphics-geek.blogspot.com/2010/02/video-animation-class-in-flex-4.html>

## Les nouveautés d'Adobe AIR 2

AIR 2 est la nouvelle version de la plate-forme d'Adobe dont le but est de fournir aux développeurs, la possibilité de créer des applications Internet riches, exécutées en dehors du navigateur.

Depuis la version initiale, il n'y a eu pas trop de nouvelles fonctionnalités importantes, mais surtout des améliorations du langage et des corrections de bugs. La version 2 a changé ça, en incluant un grand nombre de nouvelles fonctionnalités et améliorations de performances.

### Processus natifs

L'une des fonctionnalités les plus demandées par les développeurs, cette fonctionnalité permet aux applications AIR de démarrer et d'interagir avec les processus d'origine. Pour avoir accès aux processus natif, un installateur d'applications natives doit avoir été utilisé pour installer votre application au lieu de l'installateur d'applications intégré dans le moteur d'exécution AIR.

La classe `flash.desktop.NativeProcess` permet à une application AIR de lancer, d'interagir avec et de terminer un processus. Les interactions programmées sont facultatives et asynchrones et peuvent être réalisées grâce à l'utilisation de flux standard. Le processus doit être en mesure de communiquer à travers ces flux, sinon la communication ne peut être atteint.

La classe `flash.desktop.NativeProcessStartupInfo` contient des informations sur un processus destiné à être lancé. Une instance de cette classe est passée comme argument à la fonction `NativeProcess.start()`.

La classe `flash.events.NativeProcessExitEvent` est la classe d'événements standard envoyée par le processus natif quand il est quitté. Cet événement contient le code de sortie du processus, de sorte qu'il puisse être utilisé pour d'autres actions.

### Mise à jour de version de WebKit

Une nouvelle branche de Webkit, équivalente à la branche de Safari 4.0.3, a été utilisée pour construire AIR.

Les principales nouveautés sont:

- Création de versions du comportement de WebKit pour ne pas casser la compatibilité avec AIR 1.5
- Ajout de l'élément `<head>` au document s'il n'est pas présent
- Ajout du rapport d'erreur pour un appel `XMLHttpRequest` vers une ressource non-existante
- Les requêtes `XMLHttpRequest` de type `POST` ayant un longueur de 0 ne sont pas transformées en requêtes de type `GET` comme le fait AIR 1.5
- Un meilleur support de JavaScript, CSS3, l'élément `Canvas` et des données: URL.

Le nouveau moteur JavaScript (SquirrelFish Extreme) exécute les applications jusqu'à deux fois plus vite qu'avant, sans aucune modification de code.

Regardez les références de performance ci-dessous, la différence de performance entre AIR 1.5 et 2.0 est impressionnante. Les meilleurs résultats obtenus par Safari sont dues aux optimisations de taille auxquelles AIR 2 doit se conformer.

Références de performance utilisant Google V8 et WebKit SunSpider

Système d'exploitation / Référence	AIR 1.5.3	AIR 2	Safari 4.0.4
Windows XP / V8 (La plus grande valeur est la meilleure)	158.6	1157.8	1509.4
Windows XP / SunSpider (La plus petite valeur est la meilleure)	3286.4	1625.4	666.2
Mac OS X 10.6 / V8 (La plus grande valeur est la meilleure)	374.4	2522.8	2619
Mac OS X 10.6 / SunSpider (La plus petite valeur est la meilleure)	1758.8	608.2	374.4

En outre, en termes de support CSS3, AIR 2 est livré avec quelques caractéristiques notables:

- Transformation, animations et transitions 2D;
- Barres de défilement stylisable – Utilisation de propriétés CSS propriétaires pour mettre en forme et configurer les barres de défilement;
- Support du texte en colonnes – Mise en page du texte dans un conteneur à travers un nombre arbitraire de colonnes;
- Zoom – AIR 2 supporte la nouvelle propriété de zoom

## Nouvelle API réseau

AIR 2 apporte plusieurs améliorations clés, notamment les réseaux sécurisés et un serveur de socket, IPv6 et UDP.

Toutes les classes de réseaux, y compris les classes de Socket, prennent en charge les protocoles IPv4 et IPv6.

La nouvelle classe `flash.net.DatagramSocket` fournit la capacité de communiquer avec d'autres hôtes utilisant le protocole Internet UDP.

La classe `flash.net.SecureSocket` permet aux applications de se connecter avec des serveurs sécurisés utilisant SSL et TLS et d'activer une communication cryptée. Le socket est connecté uniquement si le système d'exploitation a pu vérifier qu'il n'y a pas de problèmes avec le certificat ou la chaîne de certificats.

Une application AIR peut agir comme un serveur en utilisant la classe `flash.net.ServerSocket`. En ajoutant un écouteur pour le serveur de socket, le code peut être notifié des messages quand une connexion a été établie grâce à l'événement `flash.events.ServerSocketConnectEvent`.

D'autres améliorations incluent un support pour interroger les enregistrements DNS des ressources - `Flash.net.dns.DNSResolver` et les interfaces réseau disponibles sur l'hôte - `flash.net.NetworkInfo` et `flash.net.NetworkInterface`.

## **Meilleure intégration avec le système d'exploitation**

AIR 2 vise une meilleure intégration avec le système d'exploitation et le système de fichiers grâce à certaines fonctions comme la détection des périphériques de stockage de masse, l'API d'ouverture de documents, l'amélioration de l'API d'impression et le support pour les installateurs natifs.

La gestion des dispositifs de stockage de masse est assurée par la classe `flash.filesystem.StorageVolumeInfo`. La classe représente chaque appareil disponible et accessible sur l'ordinateur et donne des informations sur les propriétés des dispositifs. L'accès est fourni dans le répertoire racine comme un objet `flash.filesystem.File`.

L'Ouverture de documents avec l'application par défaut associée au type de fichier est accompli grâce à l'utilisation de la méthode `openWithDefaultApplication` de l'objet `flash.filesystem.File`. Pour des raisons de sécurité, les types exécutables ne peuvent pas être ouverts, à moins que l'application n'ait été déployé par un installateur natif.

En termes d'impression multi-plateforme, AIR 2 a ajouté encore plus de classes à celles existantes. Les nouvelles classes sont regroupées sous le même package `flash.printing` et couvrent des fonctionnalités telles que la spécification des valeurs disponibles pour les formats de papier - classe `PaperSize`, la méthode d'impression - `PrintMethod`, et les options disponibles pour boîte de dialogue d'impression qui s'affiche à l'utilisateur - `PrintUIOptions`.

## **L'enregistrement audio local**

Avec la version 2, AIR supporte l'accès au microphone par la classe `flash.media.Microphone`. La classe permet à une application d'interroger le dispositif et de définir des propriétés différentes. En outre, il permet d'accéder aux flux sonore, afin que les données puissent être jouées ou enregistrées. Les données d'échantillonnage à la volée peuvent être obtenues en utilisant un objet `SoundTransform`.

## **Support de la gestion des gestes et du touché multiple**

Grâce à cette nouvelle fonctionnalité, une application AIR peut utiliser des gestes tels que le pincement, le défilement, la rotation, la mise à l'échelle et le touché sur deux doigts. Presque tous les composants visuels du framework Flex offre le support de gesture et multi-touch en permettant l'utilisation d'événements spéciaux. L'interrogation des dispositifs en vue de déterminer si ils sont multi-touch peut être obtenu par l'utilisation de la classe `flash.ui.Multitouch`. Les API Multi-touch et Gesture sont également disponibles avec la pré-version de AIR pour Android.

Les autres fonctionnalités notables sont:

- La gestion des exceptions globales dans le cadre de la mise à jour FlashPlayer;
- Support de lecture d'écran (Windows uniquement) - les boîtes de dialogue d'exécution peuvent être lues;
- les points de sauvegarde des transactions de base de données - la base de données SQLite inclus dans AIR fournit un support pour les points de sauvegarde au sein des transactions. La classe `flash.data.SQLConnection` offre trois nouvelles fonctions - `SQLConnection.setSavepoint ()`, `SQLConnection.releaseSavepoint ()` et `SQLConnection.rollbackToSavepoint ()` pour travailler avec des points de sauvegarde;
- File promises - une nouvelle fonctionnalité permettant la création et la manipulation de références à des fichiers qui n'existent pas encore. À un certain moment dans le processus, les fichiers doivent être créés.

Plus d'informations concernant les nouvelles fonctionnalités de AIR 2 (y compris des applications de démonstration) peuvent être trouvées aux adresses ci-dessous:

<http://www.infoq.com/articles/air-20-new-features>

[http://imageready.info/devnet/air/ajax/articles/air\\_and\\_webkit.html](http://imageready.info/devnet/air/ajax/articles/air_and_webkit.html)

[http://blogs.adobe.com/air/2010/06/adobe\\_air\\_2\\_sdk\\_now\\_available.html](http://blogs.adobe.com/air/2010/06/adobe_air_2_sdk_now_available.html)

[http://www.adobe.com/devnet/air/flex/articles/exploring\\_file\\_capabilities.html](http://www.adobe.com/devnet/air/flex/articles/exploring_file_capabilities.html)

<http://lifeonflex.blogspot.com/2010/07/see-whats-new-in-air-2-features-list.html>

<http://www.webkitchen.be/2010/02/25/whats-new-in-fp10-1-and-air2-slides-and-source-files/>

<http://www.slideshare.net/sjespers/whats-new-in-flash-player-101-and-air-2>

## Le framework Text Layout

Le framework Text Layout est une librairie au-dessus de l'API de bas niveau du moteur de texte Flash introduit par Flash Player 10. Ce framework introduit le support de l'impression de qualité du texte dans Flash (y compris le support pour les langues de droite à gauche).

Au dessus de ce framework, Spark introduit 5 nouveaux composants de texte:

- 3 composants primitifs de texte (Label, Rich Text et RichEditableText)
- 2 composants de texte personnalisables (TextInput et TextArea)

Ces composants supportent:

- Les colonnes
- Les attributs au niveau des caractères et des paragraphes
- Le crénage
- Les transformations
- Masques et modes de fusion
- Marges et indentations
- Direction (gauche à droite et droite à gauche)

### Utilisation des polices

Vous pouvez utiliser des polices de périphérique, ainsi vous n'augmentez pas la taille du fichier SWF en incorporant les polices. Toutefois, il est conseillé de spécifier à la fin de la liste des polices, une police générique. Ainsi, si par exemple Helvetica n'est pas disponible sur l'appareil, il peut utiliser d'autres polices disponibles de la famille San Serif. Voici les valeurs à choisir parmi: `_sans`, `_serif`, `_typewriter`. Vous pouvez déclarer la famille de polices comme ceci:

```
.myClass {  
    fontFamily: Arial, Helvetica, "_sans";  
}
```

Si vous choisissez d'incorporer des polices (les types de fichiers pris en charge comprennent les polices TrueType - . ttf, les polices OpenType - . otf, ainsi que les collections TrueType - . ttc, Mac Data Fork Fonts - . dfont, et Mac Resource Fork TrueType Suitcases - qui n'a pas d'extension de fichier), vous bénéficiez des avantages suivants:

- L'environnement client ne doit pas avoir la police d'installée.
- Les polices incorporées peuvent subir des rotations et des changements d'opacité.
- Les polices incorporées sont anti-aliasées, ce qui signifie que leurs bords sont lissés pour faciliter la lisibilité. Cela est particulièrement visible lorsque la taille du texte est grande.
- Les polices incorporées fournissent une lecture plus fluide lors d'un zoom.
- Le texte apparaît exactement comme vous l'attendez lorsque vous utilisez des polices intégrées.
- Lorsque vous incorporez une police, vous pouvez utiliser les informations avancées d'anti-aliasing qui offre un rendu de texte clair et de haute qualité dans les fichiers SWF. L'utilisation avancé de l'anti-aliasing améliore considérablement la lisibilité du texte, en particulier quand il est rendu dans de petites tailles de polices.

Vous incorporez une police en utilisant la déclaration CSS font-face ou la métadonnée Embed. Si vous souhaitez contrôler la taille du fichier SWF, vous pouvez utiliser des plages de caractères en incluant uniquement ceux que vous utilisez pour une déclaration spécifique font-face.

Ressources:

<http://blogs.adobe.com/flexdoc/fonts.pdf>

## Travailler avec des composants Flex 3 et 4 dans le même projet

Les composants Spark peuvent être utilisés à l'intérieur des conteneurs MX et les composants MX peuvent être utilisés à l'intérieur des conteneurs Spark.

Note: Les enfants direct d'un conteneur de navigation MX peuvent être soit des conteneurs MX, soit des conteneurs de mise en page ou de navigation, ou un conteneur Spark *NavigatorContent*. Vous ne pouvez pas imbriquer un contrôle ou un conteneur Spark autre que le conteneur Spark *NavigatorContent* dans un conteneur de navigation. Pour utiliser un conteneur Spark autre que le conteneur Spark *NavigatorContent* en tant qu'enfant d'un conteneur de navigation, vous devez l'englober dans un conteneur MX ou dans un conteneur Spark *NavigatorContent*.

## Migrer des projets Flex 3 vers Flex 4

Si vous avez besoin des fonctionnalités de Flex 4 dans un projet Flex 3, vous vous retrouvez dans la position de migrer le projet de Flex 3 à Flex 4. La procédure se fait facilement. Voici les principales étapes que vous devez réaliser:

1. Premièrement, vous devez changer l'espace de nom MXML 2006 vers les nouveaux espaces de noms

```
xmlns:fx="http://ns.adobe.com/mxml/2009"  
xmlns:mx="library://ns.adobe.com/flex/mx"  
xmlns:s="library://ns.adobe.com/flex/spark"
```

2. La balise `Declarations`. Si votre application a des composants non-visuels définis en MXML (comme par exemple `RemoteObject`, `HTTPService`, etc), vous devez les déplacer à l'intérieur de la balise

```
<fx:Declarations>
```

3. Si vous utilisez CSS, vous devez ajouter des espaces de noms pour les sélecteurs de type (`Button`, `Panel`, etc):

```
<fx:Style>  
    @namespace s "library://ns.adobe.com/flex/spark";  
    @namespace mx "library://ns.adobe.com/flex/mx";  
    s|Button {  
        fontSize:16;  
    }  
    mx|Button {  
        color:red;  
    }  
</fx:Style>
```

4. Vous devez cibler Flash Player 10 à la place du 9
5. Vous devez basculer des modèles d'états Flex 3 vers les modèles d'états Flex 4 si vous utilisez les états

## Ressources

[http://www.adobe.com/devnet/flex/articles/flex3\\_to\\_flex4\\_transitioning.html](http://www.adobe.com/devnet/flex/articles/flex3_to_flex4_transitioning.html) \\  
[http://help.adobe.com/en\\_US/Flex/4.0/FeaturesAndMigration/flex\\_4\\_features.pdf](http://help.adobe.com/en_US/Flex/4.0/FeaturesAndMigration/flex_4_features.pdf) \\  
[http://www.adobe.com/devnet/flex/articles/flexbuilder3\\_to\\_flashbuilder4.html](http://www.adobe.com/devnet/flex/articles/flexbuilder3_to_flashbuilder4.html)

## **Nouvelles fonctionnalités de Flash Builder 4 (éditeur d'états, génération de code pour les accesseurs d'attributs, les écouteurs d'événements, gestion des performances et gestion du débogage)**

Flash Builder 4 introduit un certain nombre d'améliorations qui rendent le développement d'applications Flex encore plus rapide qu'auparavant.

Support des états en vue code.

Meilleur débogueur: points d'arrêt conditionnels, allez à la ligne de commande, points d'observation, et les évaluations d'expression.

Meilleur profileur: vous pouvez vérifier si un objet sera nettoyé ou pas.

Hiérarchisation: vous pouvez afficher tous les appels à la variable sélectionnée, à une fonction / méthode, à une classe ou à une interface.

Les générateurs de code pour les gestionnaires d'événements et les accesseurs d'attributs (getter/setter)

Assistants de développement centré sur les données -support de Java, ColdFusion, PHP, Web Services, Services REST. DCD (Data Centric Development) génère le connecteur client du service et les types de données basés sur les services du serveur.

Surveillance réseau et test fonctionnels.

Réorganisation de code - déplacement et renommage.

Affichage du contenu de la documentation - Placez le curseur sur le code ou utiliser la surbrillance de code pour afficher les informations de la documentation.

Indentation du code, y compris le support pour le copier / coller.

Compilation par ligne de commande – Synchronisation individuelle des paramètres de compilation avec un environnement de compilation performant

Les tests unitaires - générer et modifier des tests FlexUnit reproductibles qui peuvent être exécutés depuis des scripts ou directement dans Flash Builder. Prise en charge de l'automatisation des composants Spark. Prise en charge de l'automatisation pour les applications Adobe AIR.

## Ressources

Un des meilleurs endroits pour obtenir des ressources sur Flex est Adobe Developer Connection et Adobe TV (où vous pouvez trouver tous les enregistrements à partir d'Adobe MAX 2008). Enfin, n'oubliez pas d'installer Tour de Flex (il s'agit d'une application AIR qui offre des tonnes d'exemples de code pour Flex 3 et Flex, y compris des composants tiers. Il est mis à jour régulièrement).

Documentation en ligne de Flex 4 et Flash Builder 4:

[http://help.adobe.com/en\\_US/flex/using/WSa7fd4845e77b4e67406b2ad31276e10ce3d-8000.html](http://help.adobe.com/en_US/flex/using/WSa7fd4845e77b4e67406b2ad31276e10ce3d-8000.html)

AdobeTV:

<http://tv.adobe.com>

Adobe Developer Connection:

<http://www.adobe.com/devnet/flex>

Tour de Flex:

<http://www.adobe.com/devnet/flex/tourdeflex/>

Les nouveautés de Flex 4:

[http://www.adobe.com/devnet/flex/articles/flex4sdk\\_whatsnew.html](http://www.adobe.com/devnet/flex/articles/flex4sdk_whatsnew.html)

Différences entre Flex 3 et Flex 4:

[http://www.adobe.com/devnet/flex/articles/flex3and4\\_differences.html](http://www.adobe.com/devnet/flex/articles/flex3and4_differences.html)

Un rapide aperçu de l'architecture Spark et du jeu de composants:

[http://www.adobe.com/devnet/flex/articles/flex4\\_sparkintro.html](http://www.adobe.com/devnet/flex/articles/flex4_sparkintro.html)

Les nouveautés de Flash Builder 4:

<http://andrewrshorten.wordpress.com/2010/03/23/whats-new-in-flash-builder-4-all-of-this/>

Migrer de Flex 3 vers Flex 4:

[http://www.adobe.com/devnet/flex/articles/flexbuilder3\\_to\\_flashbuilder4.html](http://www.adobe.com/devnet/flex/articles/flexbuilder3_to_flashbuilder4.html)

Développement centré sur les données avec Flash Builder 4:

[http://www.adobe.com/devnet/flex/articles/datacentric\\_development.html](http://www.adobe.com/devnet/flex/articles/datacentric_development.html)

Nouveau moteur d'effets:

[http://www.adobe.com/devnet/flex/articles/flex4\\_effects\\_pt1.html](http://www.adobe.com/devnet/flex/articles/flex4_effects_pt1.html)

[http://www.adobe.com/devnet/flex/articles/flex4\\_effects\\_pt2.html](http://www.adobe.com/devnet/flex/articles/flex4_effects_pt2.html)

Introduction à la personnalisation dans Flex 4:

[http://www.adobe.com/devnet/flex/articles/flex4\\_skinning.html](http://www.adobe.com/devnet/flex/articles/flex4_skinning.html)

Mise en page Spark avec Flex 4:

[http://www.adobe.com/devnet/flex/articles/spark\\_layouts.html](http://www.adobe.com/devnet/flex/articles/spark_layouts.html)