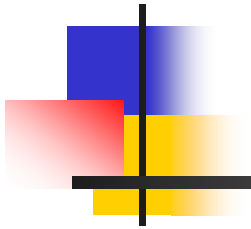
Flex

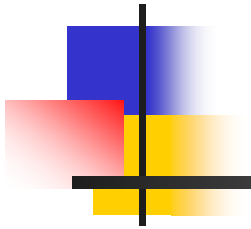
Personnaliser son application

Inspiré du livre Flex & Air



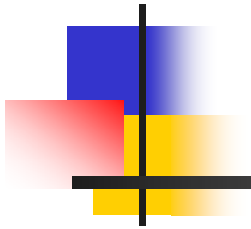
Introduction

- Objectif du chapitre
  - Découvrir comment améliorer graphiquement l'application grâce aux multiples effets graphiques ainsi qu'aux styles MXML
  - Pour cela, on peut :
    - utiliser les comportements
    - ajouter des effets
    - changer de vues (viewStates) et opérer des transitions



Personnaliser son application

- Utiliser des comportements
  - Les comportements sont des mouvements ou des animations en réponse à des actions utilisateur
  - Exemple
    - Une boîte de dialogue apparaîtra doucement comme si elle sortait de l'interface



Personnaliser son application

- Comment fonctionnent les comportements ?
  - Pour les mettre en œuvre, on utilise :
    - des triggers (similaires aux événements) et des effets
  - Un trigger (déclencheur) est
    - une action telle que cliquer sur un bouton à l'instant où un composant devient visible
  - Alors qu'un effet est
    - un changement visuel d'un composant pendant un laps de temps
      - Les effets les plus connus sont les mouvements, les redimensionnements ou les fondus

Personnaliser son application

Utiliser des comportements

- Appliquer les comportements aux composants
  - On peut le faire en MXML ou en ActionScript
  - En MXML
    - On utilise la propriété correspondante au trigger désiré, puis on lui affecte la valeur d'un effet en liant le trigger avec l'identifiant de l'effet
  - Syntaxe

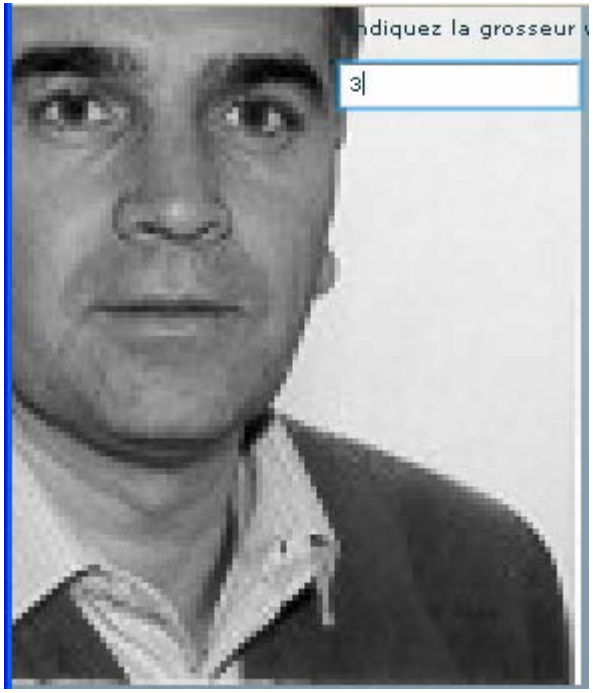
```
1. //on précise l'effet
<mx:monEffet id="monEffet" parametresEffet />
2. //on indique le composant sur lequel l'effet doit se produire
<mx:monComposant label= "monComposant"
    creationCompleteEffect="{monEffet}"/>
```

Effet	Description
AnimateProperty	Fait évoluer une propriété numérique (width, height…) fromValue toValue durant une duration
Blur	Fait apparaître du flou sur le composant
Dissolve	Fait baisser le Alpha de alphaFrom (min 0.0, totalement transparent) à alphaTo (max 1.0, totalement opaque). On peut préciser une couleur
Fade	Le contraire de Dissolve
Glow	Met un halo lumineux autour du composant
Iris	fait apparaître le composant comme un rectangle qui s'agrandit : scaleFrom, scaleTo aux dimensions scaleXTo et scaleYTo
Move	fait bouger le composant de xFrom, yFrom à xTo, yTo. Les valeurs de départ ou de fin peuvent être omises si on précise : xBy et yBy
Pause	Permet de faire une halte. Intéressant dans les interfaces où on a prévu plusieurs étapes
Resize	widthFrom, heightFrom à widthTo, heightTo. On peut utiliser widthBy et heightBy comme dans Move
SoundEffect	Joue un MP3 dont le nom est précisé dans le paramètre source
WipeLeft, WipeRight, WipeUp, WipeDown	Fait surgir le composant d'un de ces quatre côtés
zoom	Zoom de zoomHeightFrom, zoomwidthFrom à zoomHeightTo, zoomHeightFrom, zoomwidthTo

Personnaliser son application

Utiliser des comportements

- Exemple : comportement_zoom.mxml
  - On rentre la valeur du zoom d'une image dans la zone textuelle
  - En cliquant sur l'image, le zoom se réalise



Personnaliser son application

Utiliser des comportements

- Exemple : comportement_zoom.mxml

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
      //l'effet est un zoom
      <mx:Zoom id="monZoom"
      zoomHeightTo="{Number(zoomValue.text)}"
      zoomWidthTo="{Number(zoomValue.text)}"/>

      //l'effet est réalisé sur l'image en cliquant dessus
      <mx:Image source="images\photo.jpg"
                mouseDownEffect="{monZoom}"/>

      <mx:Label text="Indiquez la grosseur voulue pour
      l'image puis cliquez dessus." x="163.5" y="2"/>
      <mx:TextInput id="zoomValue"  text="" width="120"
      x="163.5" y="28"/>
      </mx:Application>
</mx:Application>
```

Personnaliser son application

Utiliser des comportements

- Comportement_blur.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
    <mx:Blur id="monWU" duration="5000"/>
    <mx:Button label="Mon Bouton"
    creationCompleteEffect="{monWU}"/>
</mx:Application>
```

Après quelques instants

Personnaliser son application

Utiliser des comportements

- Comportement_dissolve.mxml

```
//Passage du rouge vif au clair
<mx:Dissolve  color="red"  alphaFrom="0.0"  alphaTo="1.0" id
    ="monWU" />
<mx:Button label="Mon Bouton"
    creationCompleteEffect="{monWU}"/>
```

Personnaliser son application

Utiliser des comportements

- Comportement_fade.mxml

```
//Passage du pas de halot au halot
<mx:Fade id="monWU" duration="5000"/>
<mx:Button label="Mon Bouton"
    creationCompleteEffect="{monWU}"/>
```

Personnaliser son application

Utiliser des comportements

- Comportement_glow.mxml

```
<mx:Glow id="monWU" duration="5000"/>
<mx:Button label="Mon Bouton"
    creationCompleteEffect="{monWU}"/>
```

Personnaliser son application

Utiliser des comportements

- Comportement_resize.mxml

```
//Agrandissement du bouton
<mx:Resize id="monWU"   widthFrom="0"  widthTo="100"
    heightFrom="0" heightTo="100"/>
<mx:Button label="Mon Bouton"
    creationCompleteEffect="{monWU}"/>
```

Personnaliser son application

Utiliser des comportements

- Comportement_rotate.mxml

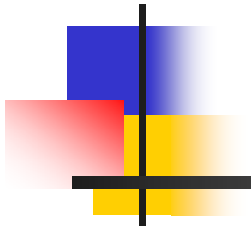
```
<mx:Rotate id="monWU"   originX="50"  originY="50"
    angleFrom="0" angleTo="90"/>
<mx:Button label="Mon Bouton"
    creationCompleteEffect="{monWU}"/>
```

Personnaliser son application

Utiliser des comportements

- Comportement_sounde.mxml

```
//Passage de son pendant 20000 millièmes de seconde
<mx:SoundEffect id="monWU" source="moliendo.mp3"
    duration="20000"/>
<mx:Button label="Mon Bouton"
    creationCompleteEffect="{monWU}"/>
```



Personnaliser son application

- Combiner plusieurs effets
  - Après avoir vu les effets simples, on va voir comment les combiner ou comment utiliser des fonctions avancées des effets
- <mx:Parallel>
  - Permet de lancer plusieurs effets en même temps
  - Exemple : effetparallel.mxml
    - rotation pendant un re-dimensionnement
    - L'image tourne en même temps qu'elle s'agrandit quand on clique dessus

Personnaliser son application

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
<mx:Parallel id="zoomAndResize">
    //Définition des deux effets
    <mx:Zoom zoomHeightTo="1.5" zoomWidthTo="1.5"/>
    <mx:Rotate />
</mx:Parallel>
  //Application de l'effet : zoomAndResize sur l'image
<mx:Image source="images\photo.jpg"
    mouseDownEffect="{zoomAndResize}"/>
</mx:Application>
```

Personnaliser son application

Ajouter des effets

- <mx:Sequence>
  - On peut appliquer plusieurs effets à un comportement les uns après les autres
  - Exemple : effetSequence.mxml
    - Cliquer sur TextArea : il s'agrandit par le milieu et ensuite bouge vers la droite

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute">
  <mx:Sequence id="irisThenMove">
      <mx:Iris/>
      <mx:Move xBy="10"/>
  </mx:Sequence>
  <mx:TextArea text="Exemple de séquence d'effets"
      mouseDownEffect="{irisThenMove}"/>
</mx:Application>
```

Personnaliser son application

Ajouter des effets

- Imbrication de séquence et parallèle : effetCombinaison.mxml

```
//En cliquant dessus, l'objet tourne (rotation), puis s'agrandit et
   se déplace (séquence)
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute">
<mx:Sequence id="combinaisonOfCombinaison">
   <mx:Parallel duration="2000">
            <mx:Iris/>
            <mx:Rotate/>
   </mx:Parallel>
   <mx:Move xBy="10"/>
</mx:Sequence>
<mx:TextArea text="Exemple de combinaison d'effets"
   mouseDownEffect="{combinaisonOfCombinaison}"/>
</mx:Application>
```

Personnaliser son application

Ajouter des effets

- Combiner avec ActionScript
  - On dispose d'un équivalent ActionScript de ce que l'on vient de voir
  - Les classes ActionScript : Parallel et Sequence disposent de la méthode addChild() qui prend en argument un objet de type Effect
  - On peut donc ajouter des effets enfants aux objets de type Parallel et Sequence et ensuite les utiliser comme on avait vu
  - Exemple
    - On va réaliser l'effet Iris qui fera apparaître le composant avec un disque qui s'agrandit
    - Tout d'abord
      - On doit fabriquer l'objet de type Shape qui sera transmis à notre effet Iris
        - Pour cela, on instancie un objet Shape puis on lui donne les caractéristiques nécessaires afin que notre disque ait le bon diamètre final, la bonne position, etc.

- Exemple : effetCombinaisonAS.mxml

```
public function createRoundMask(tag:Object,
    compDest:Rectangle):Shape
{
//on instancie l'objet
var roundMask:Shape = new Shape();
//puis on dessine un disque
roundMask.graphics.beginFill(0x00000, 0.0);
//on place notre disque de façon à le centrer sur le composant
//ainsi nous avons besoin d'avoir le centre du disque
//au centre du rectangle
roundMask.graphics.drawCircle(compDest.width/2,compDest.h
    eight/2,Math.sqrt((compDest.height*compDest.height)+(com
    pDest.width*compDest.width)));
    //La dimension du rayon est calculée grâce au théorème de
    //pythagore afin d'obtenir un cercle circonscrit au rectangle
    roundMask.graphics.endFill();
    return roundMask;
}
```

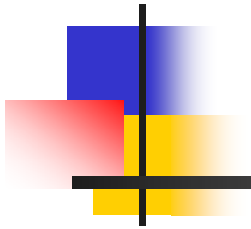
Personnaliser son application

Ajouter des effets

- Ensuite : on déclare notre effet, en déclarant quelle est la fonction qui va créer notre masque grâce à la propriété createMaskFunction

```
<mx:Iris duration="2000" id="showWL"
   createMaskFunction="createRoundMask"
   showTarget="true"/>
<mx:Image source="@Embed('images/photo.jpg')"
   mouseDownEffect="{showWL}"/>
</mx:Application>
```

En cliquant sur la photo



Personnaliser son application

- Styliser l'interface
  - On utilise la balise <mx:Style> comme dans HTML
  - Soit en interne :

```
<mx:Style>
    .monStyle{text-align :center;}
</mx:Style>
```

  - Soit en externe

```
<mx:Style source="maCSS.css"/>
```

Personnaliser son application

- Les classes de style
  - Une classe de style est un nom qui regroupe plusieurs styles
  - Exemple

```
<mx:Style>
    .maClasseStyle {
        text-align:center;
        fontSize : 10;
        color : #FFFF;
    }
</mx:Style>
```

- Exemple : css.mxml

```
<?xml version="1.0"?>
<!-- styles/ClassSelector.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Style>
        .myFontStyle {
            fontSize: 15;
            color: #9933FF;
        }
    </mx:Style>

    <!-- This button has the custom style applied to it. -->
    <mx:Button id="myButton" styleName="myFontStyle" label="Click
      Me"/>

    <!-- This button uses default styles. -->
    <mx:Button id="myButton2" label="Click Me"/>

</mx:Application>
```

- Exemple : styleBouton.mxml

```
<?xml version="1.0"?>
<!-- styles/TypeSelector.mxml -->
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Style>
        Button {
            fontSize: 15;
            color: #9933FF;
        }
    </mx:Style>

    <mx:Button id="myButton" label="Click Me"/>
    <mx:Button id="myButton2" label="Click Me"/>

</mx:Application>
```

- Exemple : classStyle.mxml

```
<?xml version="1.0"?>
<!-- styles/CamelCase.mxml -->
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Style>
      .myFontStyle {
         fontSize: 15; /* Note the camelCase. */
      }
      .myOtherFontStyle {
         font-size: 15; /* Note the hyphen. */
      }
   </mx:Style>
   <mx:Button id="myButton" styleName="myFontStyle"
    label="Click Me"/>
   <mx:Button id="myButton2"
    styleName="myOtherFontStyle" label="Click Me"/>

</mx:Application>
```

- Exemple : classesStyle.mxml

```
<?xml version="1.0"?>
<!-- styles/MultipleTypeSelector.mxml -->
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
    Button, TextInput, Label {
      fontStyle: italic;
      fontSize: 24;
    }
  </mx:Style>

  <mx:Button id="myButton" label="Click Here"/>
  <mx:Label id="l1" text="My Label"/>
  <mx:TextInput id="ti1" text="Input text here"/>
</mx:Application>
```

Personnaliser son application

Utiliser des comportements

- On peut appliquer un style suite à un événement
  - Exemple : comportement_zoom_style.mxml

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute">
   <mx:Style>
            //indique qu'au click sur Image, on produit un zoom
            Image {mouseDownEffect: monZoom;}
   </mx:Style>
   <mx:Zoom id="monZoom"
   zoomHeightTo="{Number(zoomValue.text)}"
   zoomWidthTo="{Number(zoomValue.text)}"/>
   <mx:Image source="images\photo.jpg"/>
   <mx:Label text="Indiquez le zoom voulu pour l'image, puis
   cliquez dessus" />
   <mx:TextInput id="zoomValue"  text="" />
   </mx:Application>
```

- Utilisation d’un fichier .css : cssStyle.mxml

```
/* assets/SimpleTypeSelector.css */
Button { fontStyle: italic; color: #99FF00; }

//ici on combine deux styles intérieurs et extérieur
<?xml version="1.0"?>
<!-- styles/TypeSelectorWithExternalCSS.mxml -->
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Style source="../assets/SimpleTypeSelector.css"/>
    <mx:Style>
            Button { fontSize: 15; }
    </mx:Style>
    <mx:Button id="myButton" label="Click Here"/>
</mx:Application>
```

- Exemple : mixStyle.mxml
  - On peut mixer deux styles par le nom de la balise et le styleName

```
<?xml version="1.0"?>
<!-- styles/CompoundSelectors.mxml -->
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Style>
        Label {
            fontSize: 10pt;
        }
        .myLabel {
            color: Blue;
        }
    </mx:Style>
    <mx:Label styleName="myLabel" text="This label is
    10pt Blue"/>
</mx:Application>
```

- Exemple : priorityStyle.mxml
  - Ici, pour le premier bouton, c'est le style myclass qui l'emporte

```
<?xml version="1.0"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
    .myclass {
      color: Red;
    }
    Button {
      fontSize: 10pt;
      color: Yellow;
    }
  </mx:Style>
  <mx:VBox width="500" height="200">
    <mx:Button styleName="myclass" label="Red Button"/>
    <mx:Button label="Yellow Button"/>
  </mx:VBox>
</mx:Application>
```

- Exemple : css2.mxml
  - Ici, pour on voit l'effet de l'héritage du style

```
<?xml version="1.0"?>
<!-- styles/BasicInheritance.mxml -->
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Style>
    VBox {
      color:blue
    }
  </mx:Style>

  <mx:VBox width="500" height="200">
    <mx:Label text="This is a label."/>
    <mx:Button label="Click Me"/>
  </mx:VBox>

</mx:Application>
```

- Exemple : imageStyle.mxml
  - On peut utiliser une image comme style de background

```
<?xml version="1.0"?>
<!-- styles/EmbedInCSS.mxml -->
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml">
  <mx:Style>
    .style1 {
      backgroundImage:Embed("/assets/butterfly.jpg");
    }
  </mx:Style>
  <mx:HBox>
    <mx:TextArea width="200" height="200"
     styleName="style1"/>
  </mx:HBox>
</mx:Application>
```

Personnaliser son application

Styliser l'interface

- Le styleManager
  - La classe Stylemanager permet d'atteindre les styles de toutes les instances d'une classe à l'exécution
- Exemple
  - Ici, on la combine avec setStyle pour agir sur le bord d'une image

    ```
    Style.Manager.getStyleDeclaration("Image").setStyle("Border",0)
    ```
  - De cette façon, toutes les images de l'application vont avoir leur propriété de style Border mise à 0
  - Cette classe permet de remettre à jour tous les styles d'un type de composant au lancement de l'application

Personnaliser son application

Styliser l'interface

- Le style inline
  - On peut utiliser le phénomène de liaison pour agir sur le style
  - Exemple : pour changer la couleur

```
<mx:HBox width="100" height="100" backgroundColor
    ="{couleur}"/>
```

  - permet de changer la couleur de fond en utilisant un événement :

Personnaliser son application

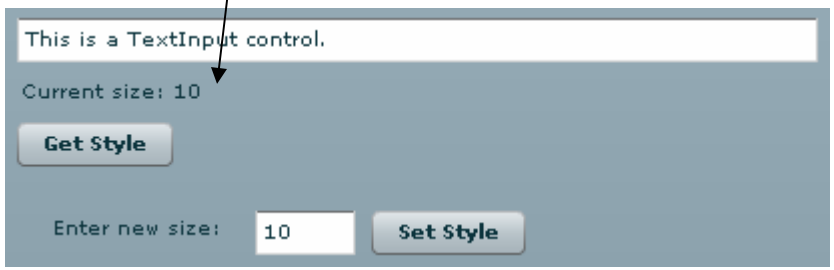
Styliser l'interface

- Exemple complet : styleInline.mxml

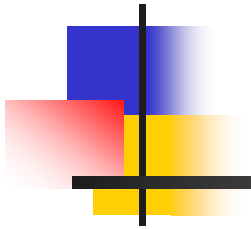
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
<mx:Script>
<![CDATA[
[Bindable]
public var color:uint= 0x0;
public function changeColor() : void
{
    color=0xFFFFFF;
}
]]>
</mx:Script>
<mx:HBox width="100" height="100" backgroundColor ="{color}"/>
<mx:Button click="changeColor()"/>
</mx:Application>
```

- On peut aussi utiliser les méthodes ActionScript :
  - setStyle() et getStyle()
  - Exemple : comportement_as.mxml

```
// la première partie MXML
<mx:VBox id="vb">
  <mx:TextInput id="ip1" styleName="myClass"  text="This is a
  TextInput control." width="400"/>
  <mx:Label id="lb1" text="Current size: {curSize}" width="400"/>
  <mx:Button id="b1" label="Get Style" click="showStyles();"/>
  <mx:Form>
   <mx:FormItem label="Enter new size:">
      <mx:HBox>
         <mx:TextInput text="{curSize}" id="ip2" width="50"/>
         <mx:Button id="b2" label="Set Style"
             click="setNewStyles();"/> // met à jour le style
      </mx:HBox>
   </mx:FormItem>
  </mx:Form>
</mx:VBox>
```

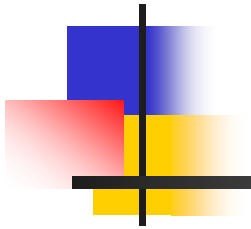


```
//Les fonctions .as appelées
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
     creationComplete="initApp()">
<mx:Script><![CDATA[
    [Bindable]
    private var curSize:int = 10;
    //initialise le style
    private function initApp():void {
        ip1.setStyle("fontSize", curSize); //fontsize est un paramètre standard de Flex
        b1.setStyle("fontSize", curSize);
        b2.setStyle("fontSize", curSize);
    }
    //affiche le style utilisé
    public function showStyles():void {
        mx.controls.Alert.show("Font size is " + ip1.getStyle("fontSize") + ".");
     }
     //modifie le style
     public function setNewStyles():void {
        curSize = Number(ip2.text);
        ip1.setStyle("fontSize", curSize);
        b1.setStyle("fontSize", curSize);
        b2.setStyle("fontSize", curSize);
     }
  ]]>
</mx:Script>
```



Les filtres en Flex

- Rôle
  - On peut utiliser les filtres Adobe Flash pour obtenir un effet de style
  - On peut appliquer des filtres à tout élément visuel Flex qui est dérivé de UIComponent
  - Les filtres sont pas des styles, car vous ne pouvez pas les appliquer avec une feuille de style ou avec la méthode setStyle ()
  - Le résultat d'un filtre, cependant, est souvent considéré comme un style

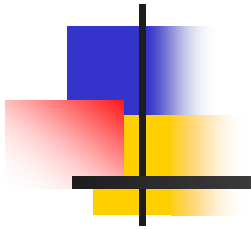


Les filtres en Flex

- Rôle (suite)
  - Les filtres sont dans le package flash.filters.* package, et incluent les classes DropShadowFilter, GlowFilter, et BlurFilter
  - Pour appliquer un filtre à un composant avec MXML, on ajoute la classe du filtre au tableau des filtres du composant
  - Le tableau des filtres est une propriété héritée dela classe DisplayObject
  - Elle contient toute sorte de filtres qu'on peut appliquer à un composant

Les filtres en Flex

- Exemple : filtre1.mxml
  - L'exemple suivant applique une ombre portée à un contrôle de type Label

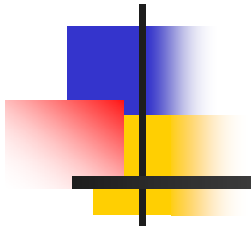


Les filtres en Flex

- Exemple : filtre1.mxml

```
<?xml version="1.0"?>
<!-- styles/ApplyFilterInline.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <!-- Apply filter using MXML syntax to set properties. -->
    <mx:Label text="DropShadowFilter" fontSize="20">
        <mx:filters>
            <mx:DropShadowFilter distance="10" angle="45"/>
        </mx:filters>
    </mx:Label>

    <!-- Apply filter and set properties inline. -->
    <mx:Label
        text="DropShadowFilter (inline)"
        fontSize="20"
        filters="{[new DropShadowFilter(10, 45)]}"
    />
</mx:Application>
```



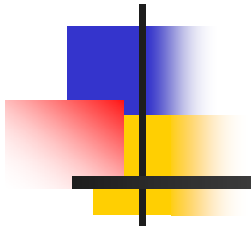
Les filtres en Flex

- Application en ActionScript
  - Vous pouvez appliquer des filtres en ActionScript
  - Pour ce faire
    - importer le package flash.filters .*
    - puis ajouter le nouveau filtre aux filtres Array du contrôle Flex
    - L'exemple suivant applique une ombre blanche au contrôle Label lorsque l'utilisateur clique sur le bouton :

Exemple : filtre2.mxml

```
<?xml version="1.0"?>
<!-- styles/ApplyFilterAS.mxml -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script><![CDATA[
        import flash.filters.*;

        public function toggleFilter():void {
            if (label1.filters.length == 0) {
                /* The first four properties of the DropShadowFilter
    constructor are
                    distance, angle, color, and alpha. */
                var f:DropShadowFilter = new
    DropShadowFilter(5,30,0xFFFFFF,.8);
                var myFilters:Array = new Array();
                myFilters.push(f);
                label1.filters = myFilters;
            } else {
                label1.filters = null;
            }
        }
    ]]></mx:Script>
    <mx:Label id="label1" text="ActionScript-applied filter."/>
    <mx:Button id="b1" label="Toggle Filter" click="toggleFilter()"/>
</mx:Application>
```

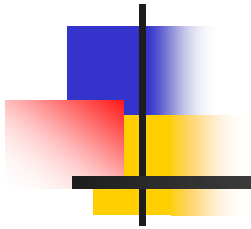


Filtres en Flex

- Exemple : filtre3.mxml
  - Si vous modifiez un filtre, vous devez l'affecter à la composante afin que les changements prennent effet.
  - L'exemple suivant modifie la couleur des filtres lorsque vous cliquez sur le bouton :

Exercice 3

- Énoncé
  - Réaliser une petite interface permettant d'agir, par interaction, sur la saturation et le alpha:



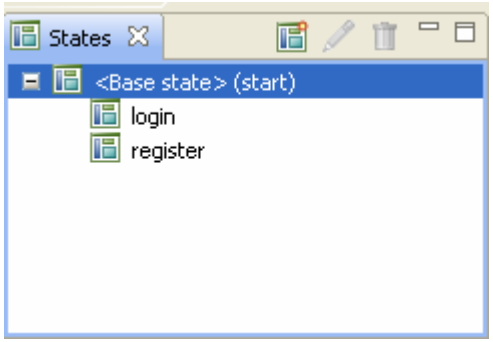
Personnaliser son application

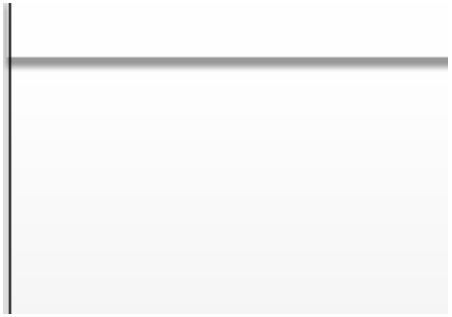
- Travailler avec des états de vue : viewStates
  - Les vues servent à créer des affichages différents de l'application
  - On démarre avec une vue de base
  - Puis, on crée des vues supplémentaires en ajoutant des éléments à la vue de base
  - Chaque vue a un nom
  - On peut créer des transitions entre les vues

Personnaliser son application

Travailler avec des états de vue

- Au départ
  - l'application est créée avec un seul état (vue) appelé : Base state
- Ensuite
  - On peut créer des états de vue
    - Clic droit sur <Base State>
- Exemple :
  - ici, on a créé 2 états : login et register

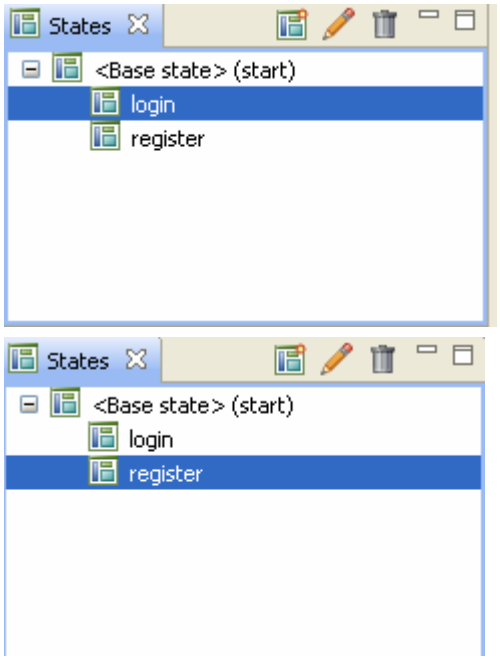




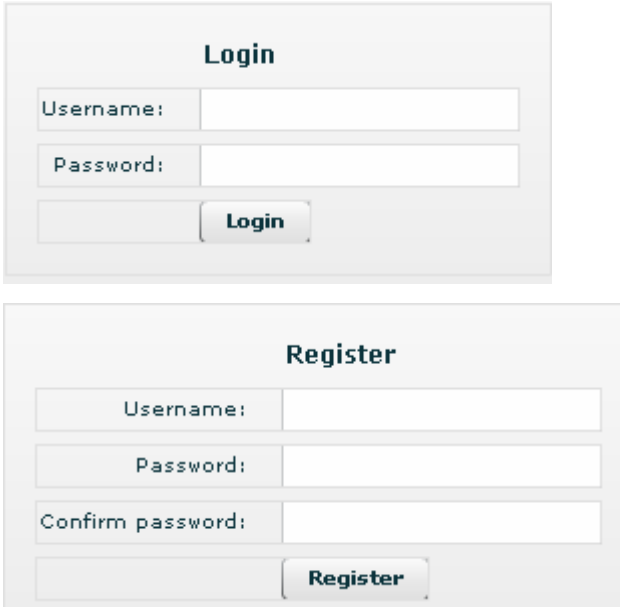
Personnaliser son application

Travailler avec des états de vue

- Les deux états correspondent à deux affichages après sélection par l'utilisateur d'une action spécifique



États



Vues correspondantes

Personnaliser son application

Travailler avec des états de vue

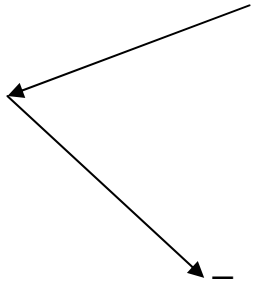
- Exemple complet : viewStates.mxml
  - Étape 1 : vue de base
    - On crée une image avec en dessous son nom et un lien pour afficher dans une autre vue les détails sur l'image

Personnaliser son application

Travailler avec des états de vue

- Étape 1 : vue de base :
- C'est un Panel qui contient une image et une ligne de commande

```
<mx:Panel id="presObj" title="Image">
    <mx:Image id="imgExp" source="lena-color.png"/>
    <mx:ControlBar>
        <mx:VBox id="vBoxInfo">
            <mx:Label text="{imgExp.source}"/>
        </mx:VBox>
    </mx:ControlBar>
</mx:Panel>
```

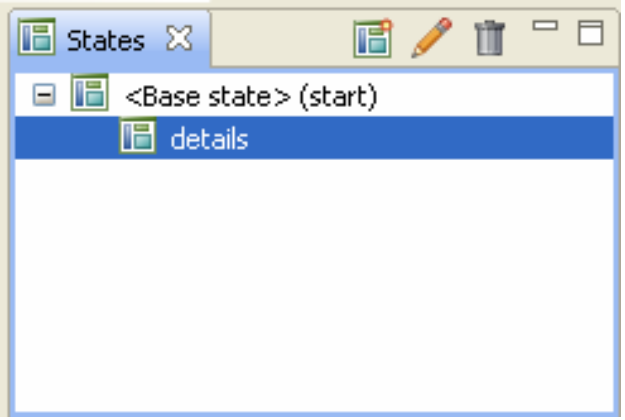


- vBoxInfo est la boîte à laquelle on a envie d'ajouter des éléments d'affichage dans la deuxième vue
- {imgExp.source} : permet d'afficher le nom du fichier image, ici : lena-color.png

Personnaliser son application

Travailler avec des états de vue

- Étape 2 : création d'une deuxième vue : "details"



Personnaliser son application

Travailler avec des états de vue

- Étape 3 : lier les deux vues
  - On ajoute en bas de l'image, dans la barre de contrôle, un lien qui va permettre de passer de
    - la vue de base : currentState=""
    - à la vue détaillée : currentState="details"

```
// ce bouton permet d'aller dans la vue details
    <mx:LinkButton label="Afficher les détails"
      id="affDetails" click="currentState='details'"/>
```

Personnaliser son application

Travailler avec des états de vue

- Étape 4 : définir ce que l'on veut faire dans la vue détaillée
  - On ajoute
    - Un label affichant les dimensions de l'image
    - Un champ de texte pour éventuellement ajouter des mots clés à l'image
    - Un lien pour revenir à la vue simple

- Chaque élément est ajouté à la nouvelle vue en
  - Le rattachant au conteneur (ici vBoxInfo), comme fils par <mx:AddChild>

```
<mx:states>
    <mx:State name="details">
        //on ajoute un fils à la vBoxInfo comprenant ces infos
        <mx:AddChild relativeTo="{vBoxInfo}" position="lastChild"
                creationPolicy="all">
                <mx:Label text="96 * 96"/>
        </mx:AddChild>
        //on ajoute un fils à la vBoxInfo comprenant ces infos
        <mx:AddChild relativeTo="{vBoxInfo}" position="lastChild"
                creationPolicy="all">
                <mx:TextInput text="mots clefs"/>
        </mx:AddChild>
        //on ajoute un fils à la vBoxInfo comprenant ces infos
        <mx:AddChild relativeTo="{vBoxInfo}" position="lastChild"
                creationPolicy="all">
                <mx:LinkButton label="Retour à  la vue simple"
                click="currentState=''"/>
        </mx:AddChild>
        <mx:SetProperty target="{presObj}" name="title" value="Détails de
        l'image"/>
    <mx:RemoveChild target="{affDetails}"/>
    </mx:State>
<mx:states>
```

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
    <mx:states>
        //ajout de l'état details
        <mx:State name="details">
            <mx:AddChild relativeTo="{vBoxInfo}" position="lastChild"
                creationPolicy="all">
                <mx:Label text="96 * 96"/>
            </mx:AddChild>
            <mx:AddChild relativeTo="{vBoxInfo}" position="lastChild"
                creationPolicy="all">
                <mx:TextInput text="mots clefs"/>
            </mx:AddChild>
            <mx:AddChild relativeTo="{vBoxInfo}" position="lastChild"
                creationPolicy="all">
                <mx:LinkButton label="Retour à  la vue simple"
                click="currentState=''"/>
            </mx:AddChild>
            <mx:SetProperty target="{presObj}" name="title"
            value="Détails de l'image"/>
            <mx:RemoveChild target="{affDetails}"/>
        </mx:State>
</mx:states>
```

```
//Définition des transitions entre les vues
<mx:transitions>
    //Cela indique que la transition de type rotation est valable de
    n'importe quelle vue à n'importe quelle vue
<mx:Transition fromState="*" toState="*">
        <mx:Rotate duration="2000" target="{imgExp}"/>
    </mx:Transition>
</mx:transitions>
<mx:Panel id="presObj" title="Image" height="300" width="300">
    <mx:Image id="imgExp" source="lena-color.png"/>
    <mx:ControlBar>
        <mx:VBox id="vBoxInfo">
            <mx:Label text="{imgExp.source}"/>
        </mx:VBox>
        <mx:LinkButton label="Afficher les détails" id="affDetails"
        click="currentState='details'"/>
    </mx:ControlBar>
</mx:Panel>

</mx:Application>
```

Personnaliser son application

Travailler avec des états de vue

- Dans l'exemple
  - On indique par <mx:SetProperty> la cible dans laquelle on veut afficher la vue, ici le Panel
  - On efface l'ancienne partie de vue (ici la controlBar) par <mx:RemoveChild>

Personnaliser son application

Travailler avec des états de vue

- Autre exemple avec des transitions : viewstates2.mxml
  - Transition de l'état de base à l'état details
  - Transition de l'état details à l'état de base