

27 septembre 2010

Tutoriel RIA Flex

1	PRISE EN MAIN DE L'ENVIRONNEMENT DE DEVELOPPEMENT	3
1.1	EXEMPLE 1, APPLICATION HELLOWORLD	4
1.2	EXEMPLE 2, APPLICATION PERROQUET	7
1.3	EXEMPLE 3, APPLICATION LOGIN	10
1.4	SKINNING DE COMPOSANTS	14
1.5	EXERCICES	19

1 Prise en main de l'environnement de développement

La prise en main de l'environnement **Adobe Flash Builder 4** débute par la création d'un nouveau projet. Pour ce faire, utiliser le menu **Fichier → Nouveau → Projet Flex**.

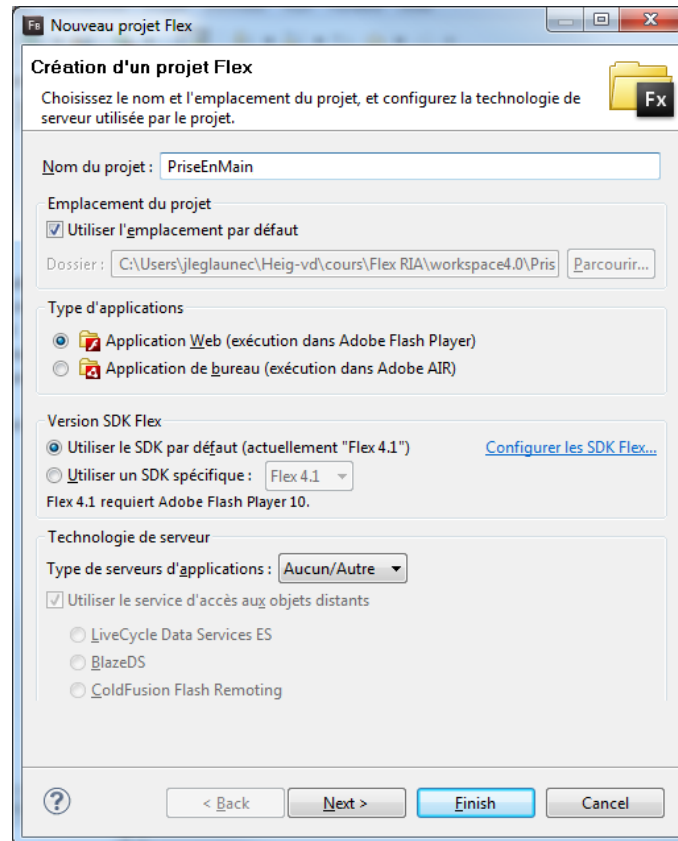








Figure 1 - Création d'un projet



Le projet créé est structuré de la manière suivante

- ▲  PriseEnMain
 - ▶  bin-debug
 - ▶  html-template
 -  libs
 - ▲  src
 -  PriseEnMain.mxml

Flash Builder 4 offre la possibilité de spécifier le thème visuel du projet. Ceci peut se faire par l'intermédiaire du menu contextuel **Properties → Thème Flex**.

1.1 Exemple 1, Application HelloWorld

Créer une application **HelloWorld** par l'intermédiaire du menu contextuel **New** → **Application MXML**.

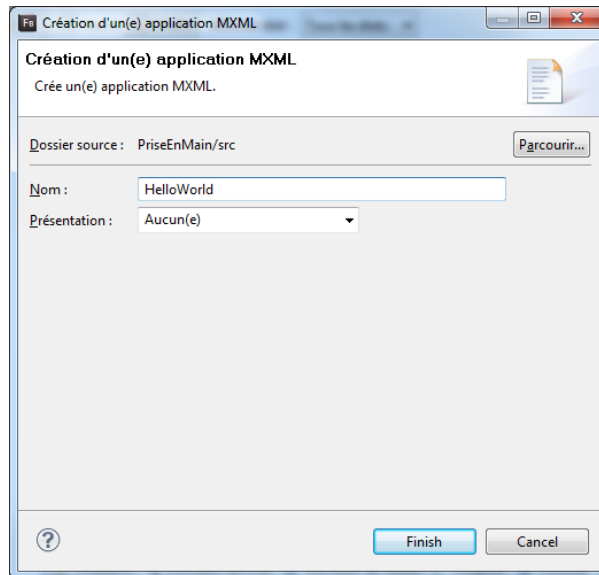


Figure 2 – Création d'une application

Le fichier suivant a été créé dans le dossier **src**

HelloWorld.xml

```
<?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"
  minWidth="955"
  minHeight="600">
  <fx:Declarations></fx:Declarations>
</s:Application>
```

1.1.1 Ajout d'un panel à l'application

La première étape consiste à afficher un panel possédant les caractéristiques suivantes :

- Titre du panel « *Hello world* »
- Layout vertical comportant une marge intérieure de 10px
- Placé au centre de l'application

Panel

```
<s:Panel
  id="pnlHelloWorld"
  title="Hello world"
  verticalCenter="1"
  horizontalCenter="1">
  <s:VGroup
    left="10"
    right="10"
    top="10"
    bottom="10">
  </s:VGroup>
</s:Panel>
```

1.1.2 Ajout d'un label

Le label possède uniquement un identifiant car son texte va être défini lors de l'événement `click` du bouton

Label

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

1.1.3 Ajout d'un bouton « Click me »

Ajouter un bouton « *Click me* » qui va se contenter d'afficher « *Hello world !* » dans le label.

Bouton

```
<s:Button
  id="btnClickMe"
  label="Click me"
  click="lblHelloWorld.text = 'Hello world!'/>
```

1.1.4 Résultat

L'application **HelloWorld** produit l'affichage suivant :



Le code source complet de l'application est disponible ci-dessous :

HelloWorld.mxml

```
<?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"
  minWidth="955"
  minHeight="600">
  <fx:Declarations></fx:Declarations>
  <s:Panel
    id="pnlHelloWorld"
    title="Hello world"
    verticalCenter="1"
    horizontalCenter="1">
    <s:VGroup
      left="10"
      right="10"
      top="10"
      bottom="10">
      <s:Button
        id="btnClickMe"
        label="Click me"
        click="lblHelloWorld.text = 'Hello world!'" />
      <s:Label
        id="lblHelloWorld" />
    </s:VGroup>
  </s:Panel>
</s:Application>
```

1.1.5 Exercice



Toujours dans notre application **HelloWorld**, afficher le message « *Application chargée* » dans une fenêtre pop-up lorsque l'interface a été entièrement chargée.

Flex Language Reference, Class **Application** :

http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/spark/components/Application.html

Flex Language Reference, Class **Alert** :

http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/mx/controllers/Alert.html?allClasses=1

1.2 Exemple 2, Application Perroquet

Le deuxième exemple consiste à créer une application **Perroquet** toujours dans le projet **PriseEnMain**. Cette application a pour but de copier la valeur d'un champ dans un autre. Un bouton est utilisé pour lire la valeur d'un champ texte et l'afficher dans un label.

La méthode suivante permet de copier la valeur d'un champ texte dans un label :

```
private function btnRepeat_click(event : MouseEvent) : void
{
    lblDestination.text = txtSource.text
}
```



Il existe différentes manières d'introduire du code ActionScript dans une application Flex.

1.2.1 Association directe de code ActionScript

```
click="lblHelloWorld.text = 'Hello world!'"
```

Cette méthode peut s'avérer utile lorsque le traitement à effectuer est composé d'une seule commande mais réduit la lisibilité lorsque le code à exécuter est plus conséquent. On l'utilise plus couramment pour appeler une méthode d'un script.

1.2.2 Association avec fonction (fx:Script)

```
<fx:Script>
  <![CDATA[
    import mx.controls.Alert;
    public function sayHelloWorld() : void {
      Alert.show("Hello world!");
    }
  ]]>
</fx:Script>
```

Cette méthode permet de centraliser le code ActionScript dans un fichier MXML.

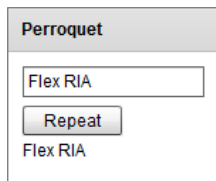
1.2.3 Importation d'un fichier ActionScript externe

```
<fx:Script source="myAsFile.as"/>
```

Cette méthode permet de séparer le code ActionScript du fichier MXML, ceci pour une séparation de la logique et de la présentation tout en permettant la réutilisation du fichier ActionScript.

1.2.4 Résultat

L'application **Perroquet** produit l'affichage suivant :



Le code source complet est disponible ci-dessous :

Perroquet.mxml

```
<?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"
               minWidth="955"
               minHeight="600">
  <fx:Script>
    <![CDATA[
      private function btnRepeat_click(event : MouseEvent) : void
      {
        lblDestination.text = txtSource.text
      }
    ]]>
  </fx:Script>
  <!-- Panel principal -->
  <s:Panel
    id="pnlPerroquet"
    title="Perroquet"
    verticalCenter="1"
    horizontalCenter="1">
    <s:VGroup
      left="10"
      right="10"
      top="10"
      bottom="10">
      <!-- Champ contenant le texte à copier -->
      <s:TextInput
        id="txtSource"/>
      <!-- Bouton permettant d'effectuer la copie -->
      <s:Button
        id="btnRepeat"
        label="Repeat"
        click="btnRepeat_click(event)"/>
      <!-- Label utilisé dans le but d'afficher le texte copié -->
      <s:Label
        id="lblDestination"/>
    </s:VGroup>
  </s:Panel>
</s:Application>
```


1.2.5 Exercice



Modifier l'application *Perroquet* afin que la copie ne se fasse plus lors du clic sur le bouton mais « *on-the-fly* » à chaque modification du champ texte `txtSource`. De plus, le texte copié sera converti en majuscules.

Flex Language Reference, Class **TextInput** :

http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/spark/components/TextInput.html

1.3 Exemple 3, Application Login

Le troisième exemple de cette prise en main consiste en une application mettant en œuvre un formulaire d'authentification. Il faut commencer par créer une application **Login** dans le projet **PriseEnMain**. Le but est ici de créer un formulaire de login simple en introduisant les notions d'états et de transitions.



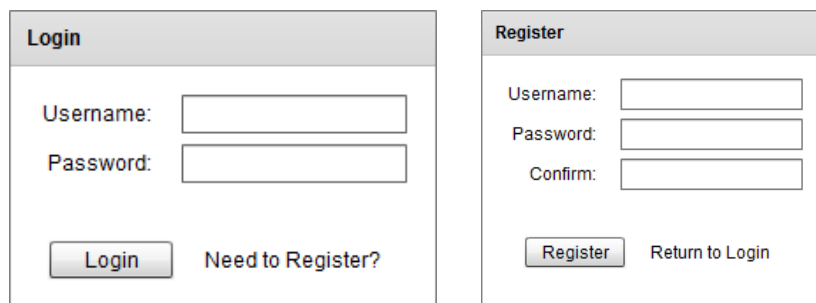
L'utilisation des états permet de dynamiser le comportement des composants visuels. En les couplant avec les transitions, ils permettent de créer des interfaces utilisateurs plus robustes et attrayantes.

Pour plus d'informations à propos des états, se référer à l'adresse suivante :

```
http://help.adobe.com/en_US/flex/using/WS2db454920e96a9e51e63e3d11c0bf690  
84-7fb4.html
```

1.3.1 Création d'un formulaire d'authentification

Cette étape consiste à créer un formulaire d'authentification à partir duquel l'utilisateur a la possibilité de s'enregistrer. Les deux actions se font par le même composant visuel en utilisant plusieurs états.



State	Fields	Buttons/Links
Login	Username: <input type="text"/> Password: <input type="password"/>	Login, Need to Register?
Register	Username: <input type="text"/> Password: <input type="password"/> Confirm: <input type="password"/>	Register, Return to Login

L'état initial du formulaire permet de s'authentifier. Pour permettre à l'utilisateur de s'enregistrer, il est nécessaire de cliquer sur le lien « *Need to Register* » ce qui provoque :

- Le remplacement du titre du panel « Login » par « Register »
- L'ajout d'un champ texte « Confirm »
- Le remplacement du label du bouton « Login » par « Register »
- Le remplacement du lien « Need to Register ? » par « Return to Login »

La création des états avec Flex 4 a été relativement simplifiée par rapport à la version précédente. Dans un premier temps, il est nécessaire de déclarer les différents états du composant visuel :

```
<s:states>  
  <s:State name="Login"/>  
  <s:State name="Register"/>  
</s:states>
```

Il est ensuite possible de déterminer plusieurs fois la même propriété d'un composant visuel. L'exemple suivant met en œuvre un panel qui n'aura pas le même titre selon l'état courant de l'interface:

```
<s:Panel
  id="loginPanel"
  title.Login="Login"
  title.Register="Register"
```

Nous pouvons également déterminer qu'un composant sera ajouté à l'interface uniquement lorsqu'on se trouve dans un état spécifique avec la propriété `includeIn`:

```
<mx:FormItem
  id="confirm"
  label="Confirm:"
  includeIn="Register">
```

1.3.2 Ajout des transitions

Les **transitions** Flex sont utilisées lorsqu'il s'agit de dynamiser une interface. Une transition s'effectue lors du passage d'un état à un autre et s'applique sur un ou plusieurs objet(s).

Transitions

```
<s:transitions>
  <s:Transition fromState="Login" toState="Register">
    <s:Wipe target="{confirm}" duration="300" direction="right"/>
  </s:Transition>
  <s:Transition fromState="Register" toState="Login">
    <s:Sequence>
      <s:Wipe target="{confirm}" duration="300" direction="left"/>
      <s:Resize target="{loginPanel}" duration="100"/>
    </s:Sequence>
  </s:Transition>
</s:transitions>
```



Il est possible de personnaliser les transitions en y ajoutant un certain nombre d'effets comme par exemple Blur, Move, Fade, Dissolve, Glow, Resize, Rotate, Zoom, etc.

Pour plus d'informations à propos des effets et transitions, se référer aux adresses suivantes :

<http://www.flex-tutorial.fr/2009/03/06/flex-4-les-nouveaux-effets-flex-de-gumbo/>

http://help.adobe.com/en_US/flex/using/WS4809A78C-9738-465d-B875-B0049C9B0ED4.html

1.3.3 Résultat

L'application **Login** produit les affichages suivants :

Login	Register
Username: <input type="text"/> Password: <input type="password"/> <input type="button" value="Login"/> Need to Register?	Username: <input type="text"/> Password: <input type="password"/> Confirm: <input type="password"/> <input type="button" value="Register"/> Return to Login

Le code source complet est disponible ci-dessous :

Login.mxml

```
<?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"
  minWidth="955"
  minHeight="600">

  <!-- Définition des états -->
  <s:states>
    <s:State name="Login"/>
    <s:State name="Register"/>
  </s:states>

  <!-- Définition des transitions -->
  <s:transitions>
    <s:Transition fromState="Login" toState="Register">
      <s:Wipe target="{confirm}" duration="300" direction="right"/>
    </s:Transition>
    <s:Transition fromState="Register" toState="Login">
      <s:Sequence>
        <s:Wipe target="{confirm}" duration="300"
direction="left"/>
        <s:Resize target="{loginPanel}" duration="100"/>
      </s:Sequence>
    </s:Transition>
  </s:transitions>

  <!-- Panel principal -->
  <s:Panel
    id="loginPanel"
    title.Login="Login"
    title.Register="Register"
    verticalCenter="1"
    horizontalCenter="1"
    height.Login="170"
    height.Register="200">

    <s:VGroup>
      <!-- Formulaire de login -->
      <mx:Form
        id="loginRegisterForm">
        <mx:FormItem
          label="Username:">
          <s:TextInput
            id="txtUsername"/>
        </mx:FormItem>
      </mx:Form>
    </s:VGroup>
  </s:Panel>
</s:Application>
```

```

        </mx:FormItem>
        <mx:FormItem
        label="Password:">
        <s:TextInput
            id="txtPassword"
            displayAsPassword="true"/>
        </mx:FormItem>
        <mx:FormItem
        id="confirm"
        label="Confirm:"
        includeIn="Register">
        <s:TextInput
            displayAsPassword="true"/>
        </mx:FormItem>
    </mx:Form>

    <!-- Barre de contrôle (Login/Register) -->
    <mx:ControlBar
        id="cbrControls"
        horizontalAlign="right" width="100%">
        <s:Button
            id="btnLoginRegister"
            label.Login="Login"
            label.Register="Register"/>
        <mx:LinkButton
            id="lnkLoginRegister"
            label.Login="Need to Register?"
            click.Login="{currentState='Register'}"
            label.Register="Return to Login"
            click.Register="{currentState='Login'}"/>
        </mx:ControlBar>
    </s:VGroup>
</s:Panel>
</s:Application>

```

1.3.4 Exercices



Modifier l'application **Login** afin de vérifier lors du clic sur le bouton « *Login* » que le « *Username = root* » et le « *Password = 1234* ». Afficher un message à l'utilisateur indiquant si l'authentification est réussie.

Flex Language Reference, Class **TextInput** :

http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/spark/components/TextInput.html



Ajouter un état à l'application **Login** qui permet de faire apparaître un label « *Bad Login* » lorsque l'authentification a échoué. De plus, ajouter un effet sur l'apparition du « *Bad Login* ».

Flex Language Reference, Class **State** :

http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/mx/states/State.html

1.4 Skinning de composants

Ce chapitre est axé sur la composante visuelle des interfaces « Rich ». Le SDK Flex 4 a pour objectif d'établir une séparation entre l'aspect visuel d'un composant et son comportement. Prenons comme exemple la classe `spark.components.Button` : Cette classe contient uniquement la logique liée au comportement d'un bouton. Les aspects visuels quant à eux sont définis dans la classe d'habillage `spark.skins.spark.ButtonSkin`.

1.4.1 Skinning de bouton

Nous allons commencer par créer un habillage de bouton simple (texte placé dans un rectangle arrondi). Pour ce faire, créer un « *composant MXML* » que l'on appellera « *ButtonSkin.mxml* ».

Le composant en question est un **Skin** applicable au composant **Button** :

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

Il doit obligatoirement implémenter les 4 états suivants :

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



La documentation définit les états à implémenter :

```
http://help.adobe.com/en\_US/FlashPlatform/reference/actionscript/3/spark/components/Button.html#SkinStateSummary
```

Nous allons maintenant utiliser la primitive `spark.primitives.Rect` afin de dessiner un rectangle. Un dégradé est utilisé pour la couleur de fond. On peut remarquer que le gradient est modifié en fonction de l'état du bouton.

```
<s:Rect id="r1" radiusX="4" radiusY="4" width="100%" height="100%">
  <s:fill>
    <s:LinearGradient rotation="90">
      <s:GradientEntry
        id="fillColorTop"
        color="0x393939"
        color.over="0x131313"
        color.down="0x000000"
        ratio="0"
        alpha="1"/>
      <s:GradientEntry
        id="fillColorBottom"
```

```
        color="0x131313"  
        color.over="0x393939"  
        color.down="0x000000"  
        ratio="1"  
        alpha="1"/>  
    </s:LinearGradient>  
</s:fill>  
</s:Rect>
```

Il est encore nécessaire d'ajouter un label permettant d'afficher le texte du bouton :

```
<s:Label  
    text="{hostComponent.label}"  
    color="0xD8D8D8"  
    textAlign="center"  
    verticalAlign="middle"  
    horizontalCenter="0"  
    verticalCenter="1" />
```



hostComponent.label permet à notre classe d'habillage d'accéder à la valeur de la propriété label du composant Button.

Créer ensuite l'application **UseMyButtonSkin.mxml** afin de mettre en œuvre un bouton utilisant le skin précédemment créé.

UseMyButtonSkin.mxml

```
<?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:VGroup  
        verticalCenter="1"  
        horizontalCenter="1">  
        <!-- Permet d'activer/désactiver le bouton -->  
        <s:CheckBox  
            id="cbx"  
            label="disable Button"/>  
  
        <!-- Bouton personnalisé-->  
        <s:Button  
            id="btn"  
            skinClass="skin.ButtonSkin"  
            width="200"  
            height="60"  
            label="Button"  
            enabled="{!cbx.selected}"/>  
    </s:VGroup>  
</s:Application>
```



skinClass="skin.ButtonSkin" permet de spécifier le skin à utiliser pour le bouton en question.

1.4.2 Résultat

L'application produit l'affichage suivant :



Le code source complet du fichier **ButtonSkin.mxml** est disponible ci-dessous :

skin.ButtonSkin.mxml



```
<?xml version="1.0" encoding="utf-8"?>
<s:Skin xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/mx"
  alpha.disabled=".5">
  <fx:Metadata>
    [HostComponent("spark.components.Button")]
  </fx:Metadata>
  <s:states>
    <s:State name="up" />
    <s:State name="over" />
    <s:State name="down" />
    <s:State name="disabled" />
  </s:states>
  <s:Rect id="r1" radiusX="4" radiusY="4" width="100%" height="100%">
    <s:fill>
      <s:LinearGradient rotation="90">
        <s:GradientEntry
          id="fillColorTop"
          color.up="0x393939"
          color.over="0x131313"
          color.down="0x000000"
          ratio="0"
          alpha="1"/>
        <s:GradientEntry
          id="fillColorBottom"
          color.up="0x131313"
          color.over="0x393939"
          color.down="0x000000"
          ratio="1"
          alpha="1"/>
      </s:LinearGradient>
    </s:fill>
  </s:Rect>
  <s:Label
    text="{hostComponent.label}"
    color="0xD8D8D8"
    textAlign="center"
    verticalAlign="middle"
    horizontalCenter="0"
    verticalCenter="1" />
</s:Skin>
```


1.4.3 Skinning de cases à cocher

L'objectif est de créer des cases à cocher différentes des traditionnelles CheckBox que l'on rencontre dans la plupart des interfaces. Pour ce faire, nous allons commencer par créer l'application que l'on appellera « *UseMyCheckboxSkin.mxml* ».

UseMyCheckboxSkin.mxml

```
<?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:HGroup gap="25" horizontalCenter="0" verticalCenter="0">
    <s:VGroup gap="5">
      <s:Label text="R currence :" paddingBottom="4"
        fontSize="11" textDecoration="underline" />
      <s:CheckBox label="Lundi" selected="true"
        skinClass="skin.CheckboxSkin"/>
      <s:CheckBox label="Mardi"
        skinClass="skin.CheckboxSkin"/>
      <s:CheckBox label="Mercredi" selected="true"
        skinClass="skin.CheckboxSkin"/>
      <s:CheckBox label="Jeudi" selected="true"
        skinClass="skin.CheckboxSkin"/>
      <s:CheckBox label="Vendredi"
        skinClass="skin.CheckboxSkin"/>
      <s:CheckBox label="Samedi"
        skinClass="skin.CheckboxSkin"/>
      <s:CheckBox label="Dimanche"
        skinClass="skin.CheckboxSkin"/>
    </s:VGroup>
  </s:HGroup>
</s:Application>
```

Cr ons maintenant le skin « *CheckboxSkin.xml* » applicable au composant `spark.components.CheckBox`. Il s'occupe de dessiner les symboles  et . La repr sentation de la case   cocher est d compos e en plusieurs  tapes :

- Affichage du symbole   partir d'une collection de segments (primitive `spark.primitive.Path`)
- Affichage d'une ombre   l'aide du filtre `spark.filters.DropShadowFilter`
- Affichage du label associ 
- Ajout d'une zone cliquable invisible au dessus du symbole

skin.CheckboxSkin.mxml

```
<?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">
  <fx:Metadata>
    [HostComponent("spark.components.CheckBox")]
  </fx:Metadata>
  <s:states>
```

```

<s:State name="up" stateGroups="unchecked" />
<s:State name="over" stateGroups="unchecked" />
<s:State name="down" stateGroups="unchecked" />
<s:State name="disabled" stateGroups="unchecked" />
<s:State name="upAndSelected" stateGroups="checked" />
<s:State name="overAndSelected" stateGroups="checked" />
<s:State name="downAndSelected" stateGroups="checked" />
<s:State name="disabledAndSelected" stateGroups="checked" />
</s:states>

<s:Group
  id="marks"
  verticalCenter="0"
  width="11"
  height="11"
  left="0">

  <!-- Checked-->
  <s:Path
    id="pthChecked"
    includeIn="checked"
    horizontalCenter="0"
    verticalCenter="0"
    width="11"
    height="11"
    data="M 100 0 C 75.148 24.853 46.191 87.574 46.191 87.574 C
46.191 87.574 14.204 40.716 0 40.716 L 25.11 41.012 L 43.787 62.213 L
79.29 0 L 100 0 Z" >
    <s:fill>
      <s:SolidColor color="0x70d000"/>
    </s:fill>
  </s:Path>

  <!-- Unchecked -->
  <s:Path
    id="pthUnchecked"
    includeIn="unchecked"
    horizontalCenter="0"
    verticalCenter="0"
    width="9"
    height="9"
    data="M 100 90.29 L 60.694 42.28 C 72.2 26.205 84.896 9.838
95.214 0 L 74.28 0.1 L 51.336 30.851 L 26.922 1.031 L 0 1.031 C 13.126
7.917 29.115 24.561 43.297 41.625 L 7.425 89.702 L 28.995 89.617 C 28.995
89.617 39.309 73.04 52.832 53.468 C 68.081 72.987 79.403 90.131 79.403
90.131 L 100 90.29 Z" >
    <s:fill>
      <s:SolidColor color="0xe83800"/>
    </s:fill>
  </s:Path>

  <!-- Zone de clic (invisible) -->
  <s:Group
    width="100%"
    height="100%"
    alpha="0">
    <s:Rect
      width="100%"
      height="100%">
      <s:fill>
        <s:SolidColor color="#FFFFFF"/>
      </s:fill>
    </s:Rect>
  </s:Group>

```

```
<!-- Ajout d'une ombre -->
<s:filters>
  <s:DropShadowFilter
    distance="1"
    strength="0.75"
    blurX="1"
    blurY="1" />
</s:filters>
</s:Group>

<!-- Label -->
<s:Label
  text="{hostComponent.label}"
  textAlign="start"
  color.over="#CCCCCC"
  color.down="#000000"
  color.overAndSelected="#CCCCCC"
  color.downAndSelected="#000000"
  left="16" right="0"
  top="3" bottom="3"
  verticalCenter="2"/>
</s:SparkSkin>
```

1.5 Exercices



Appliquer le « *ButtonSkin* » à l'interface de login créée précédemment.



Modifier ce « *ButtonSkin* » afin d'ajouter un effet « *Fade* » pour toute transition vers l'état « disabled »

Spark Property effects:

http://help.adobe.com/en_US/flex/using/WS6461B3EC-1AED-486d-A4B8-8EBD1993CE22.html