Flex

Aller vers une interface riche

Aller vers une interface riche

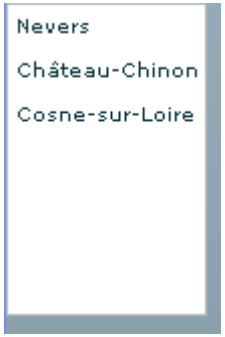
- Introduction
  - Flex offre en plus des composants de base (vus) des contrôles avancés, se prêtant à l'enrichissement et à la personnalisation
  - Parmi les contrôles, on trouve
    - List, HorizontalList, ComboBox, Tree, DataGrid de la classe ListBase
  - Ces composants sont dirigés par les données qu'ils sont chargés d'afficher :
    - C'est-à-dire qu'ils reçoivent les données à travers un provider spécifique qui conduit à une interprétation différente des données

Les contrôles avancés

- List, HorizontalList, ComboBox
  - sont des listes avec des positionnements d'éléments différents
    - <mx:List>
      - dispose les éléments de façon verticale avec une barre de défilement latéral
    - <mx:HorizontalList>
      - dispose les éléments horizontalement
    - <mx:ComboBox>
      - crée une liste déroulante

Les contrôles avancés

- Cas 1 : remplir une liste avec des chaînes de caractères : List.mxml
  - On utilise dataProvider pour spécifier ce remplissage




```
<mx:List id="ville_nievre">
   <mx:dataProvider>
         <mx:String>Nevers</mx:String>
         <mx:String>Château-Chinon</mx:String>
         <mx:String>Cosne-sur-Loire</mx:String>
   </mx:dataProvider>
</mx:List>
```

  - Ce dataprovider permet d'interpréter la liste comme un tableau

    ➔ "Nevers" aura l'index 0, "Cosne-sur-Loire ", l'index 2…

Remplissage de List, HorizontalList

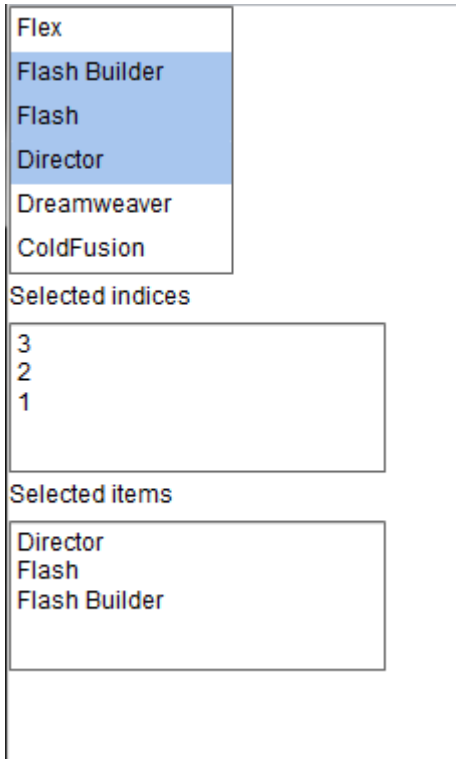
- On utilise la propriété selectedItem pour accéder aux éléments de la liste : selectedItem.mxml


```
<mx:Script>
  <![CDATA[
        import mx.controls.Alert;
        public function printData():void {
        Alert.show("La ville sélectionnée par l'utilisateur
        est "+ ville_nievre.selectedItem.data);}
  ]]>
  </mx:Script>
  <mx:List id="ville_nievre" click="printData()">
        <mx:Object label="Ne" data="Nevers"/>
        <mx:Object label="CC" data="Château-Chinon"/>
        <mx:Object label="CL" data="Cosne-sur-Loire"/>
</mx:List>
```

- Sélection multiple dans List
  - Mettre la propriété allowMultipleSelection à true
  - La valeur par défaut est : false
  - Pour sélectionner plusieurs éléments
    - Sélectionner le premier
    - Puis glissez la souris sur les suivants
    - On peut également utiliser le shift pour la sélection
  - Le contrôle List vous procure 2 vecteurs
    - selectedIndices (d'entiers) et selectedItems (d'objets)
    - Attention, les indices sont inversés : dernier sélectionné, premier indicé

http://help.adobe.com/en_US/flex/using/WSc2368ca491e3ff923c946c5112135c8ee9e-7fff.html

- Exemple : list1.mxml



http://help.adobe.com/en_US/flex/using/WSc2368ca491e3ff923c946c5112135c8ee9e-7fff.html

- Exemple : list1.mxml : la partie MXML (List)

```
<s:List allowMultipleSelection="true"
        change="myChangedHandler(event);">
    <mx:ArrayCollection>
      <fx:String>Flex</fx:String>
      <fx:String>Flash Builder</fx:String>
      <fx:String>Flash</fx:String>
      <fx:String>Director</fx:String>
      <fx:String>Dreamweaver</fx:String>
      <fx:String>ColdFusion</fx:String>
    </mx:ArrayCollection>
</s:List>

<s:Label text="Selected indices"/>
<s:TextArea id="selIndicesTA" height="75"/>
<s:Label text="Selected items"/>
<s:TextArea id="selItemsTA" height="75"/>
```

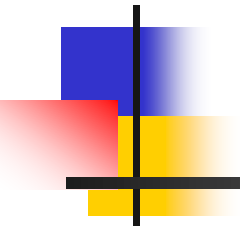
- Exemple : list1.mxml : la partie ActionScript

```
<fx:Script>
<![CDATA[
import spark.events.IndexChangeEvent;

private function myChangedHandler(event:IndexChangeEvent):void {
    var selIndices:Vector.<int> = event.currentTarget.selectedIndices;
    var selItems:Vector.<Object> = event.currentTarget.selectedItems;
    var numItems:Number = selIndices.length;
    selIndicesTA.text = "";
    selItemsTA.text = "";

    for (var i:Number = 0; i<numItems; i++){
      selIndicesTA.text = selIndicesTA.text + selIndices[i] + "\n";
      selItemsTA.text = selItemsTA.text + selItems[i] + "\n";}
}
]]>
</fx:Script>
```

Les contrôles avancés

- Cas 2 : remplir une HorizontalList avec des images : listImages.mxml
  - dataprovider sera ici un tableau d'images
  - Attention, pour que ces images soient visibles dans la liste, il faut préciser un "moteur de rendu", spécifique au type de données à afficher, ici : itemRenderer="mx.controls.Image"
  - mx.controls.Image : permet d'importer des images de différents formats

```
<mx:Script>
   <![CDATA[
            import mx.collections.*;
            import mx.controls.Image;
            private var imgArray:ArrayCollection;

            //On initialise le tableau cat avec des images
            private var cat:Array = ["images/Alien1.jpg",
            "images/Alien2.bmp", "images/Balloon.bmp", "images/Bear.jpg"];

            //On remplit la liste imgList à partir du tableau cat
            public function initImgList(items:Array):void {
                  //on tranforme l'Array cat en ArrayCollection imgArray, pour
                 // rendre possible le dataProvider
                  imgArray = new ArrayCollection(items);
                  //on range le tableau des images sous la forme d'une liste
                  imgList.dataProvider = imgArray;
            }
   ]]>
</mx:Script>

<mx:HorizontalList id="imgList" columnWidth="50" rowHeight="50"
     columnCount="4" itemRenderer="mx.controls.Image"
     creationComplete="initImgList(cat)"
/>
```



List

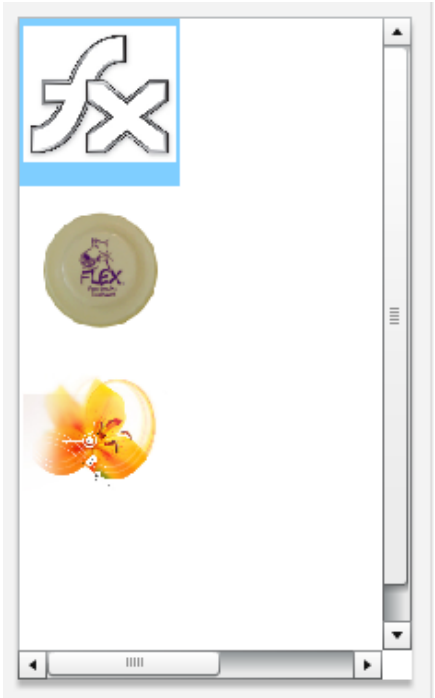
- TileList : gallerie.mxml
  - Agence ses enfants sur plusieurs lignes ou colonnes automatiquement
  - On voudrait utiliser TileList pour afficher une galerie d'images
  - Le provider de TileList est ici une liste d'images récupérées d'un fichier XML : myXML.xml
  - Le moteur de rendu des images conduit à des images :
    - verticales (rangées dans VBox) et cliquables
    - Quand on clique sur une image, on affiche l'image correspondante, plus grande
    - L'affichage se fait dans un Canvas

//Voici le fichier XML contenant les URL des images petites et grandes :
MyXml.xml

```
<chapitres>
        <chapitre>
            <titre>Image1</titre>
            <description>Ma premiere image</description>
            <image url="http://www.xarald.com/ss_domaine/ba/flex/flex_logo.jpg"
             urlImgGrand="http://www.xarald.com/ss_domaine/ba/flex/flex_logo.jpg"/>
        </chapitre>
        <chapitre>
            <titre>Image2</titre>
            <description>Ma seconde image</description>
            <image url="http://www.highflyshop.ch/catalog/images/dog-fastback-flex.gif"
            urlImgGrand="http://www.highflyshop.ch/catalog/images/dog-fastback- flex.gif"/>
        </chapitre>
        <chapitre>
            <titre>Image3</titre>
            <description>Ma troisième image</description>
            <image url="http://www.peax-webdesign.com/images/graphiste_flash.jpg"
            urlImgGrand="http://www.peax-webdesign.com/images/graphiste_flash.jpg"
            />
        </chapitre>
</chapitres>
```

List

- TileList



Le Panel des petites images cliquables

Le canevas de l'image zoomée

```
<mx:Application …>

<mx:XML id="chapitres" source="myXml.xml" />

 <mx:Panel x="0" y="91" width="234" height="400">
   <mx:TileList direction="vertical" dataProvider="{chapitres.chapitre.image}">
    <mx:itemRenderer> //on explique le rendu
        <mx:Component>
           <mx:VBox creationComplete="init()">
              //On rend les images cliquables pour les agrandir
              <mx:Image click="loadImg()" source="{data.@url}" width="83"
              height="83" />
           </mx:VBox>
        </mx:Component>
      </mx:itemRenderer>
   </mx:TileList>
 </mx:Panel>
//Affichage des grandes images
 <mx:Canvas x="500" width="500" height="400" id="canvas">
    <mx:Image id="img_grand" width="100%" height="100%" />
 </mx:Canvas>

</mx:Application>
```

Attribut de l'image

Panel

Canvas

```
//Les fonctions ActionScript
<mx:Script>
    <![CDATA[
        import mx.events.CalendarLayoutChangeEvent;
        [Bindable]
        private var urlGdImg : String;

        import mx.core.Application;
        import flash.display.DisplayObject;
        import mx.containers.Canvas
        import mx.controls.Image;

        //on initialise l'URL : urlgImg à une grande image
         private function init():void{
                urlGdImg = data.@urlImgGrand;
         }
        //On pointe le Canevas de l'application et on charge dedans (dans l'emplacement
        prévu = « img_grand ») une image dont l'URL a été sélectionnée
        private function loadImg():void{
            var c : Canvas =
            Application(Application.application).getChildByName("canvas") as Canvas;
            var img : Image = c.getChildByName("img_grand") as Image;
            img.load(urlGdImg);
        }
        ]]>
</mx:Script>
```

Permet de récupérer la racine de l'arbre

```
//Exemple en Flex 4.5
<s:Application … initialize="initData();" >

<fx:Script>
<![CDATA[
import mx.controls.Button;
import mx.collections.*;

private var listArray:Array=[
      {label: "item0", data: 0},{label: "item1", data: 1},
      {label: "item2", data: 2},{label: "item3", data: 3},
      {label: "item4", data: 4},{label: "item5", data: 5},
      {label: "item6", data: 6},{label: "item7", data: 7},
      {label: "item8", data: 8}];

[Bindable]
public var TileListdp:ArrayList;

      private function initData():void {
            TileListdp = new ArrayList(listArray);
      }
]]>
</fx:Script>
```

```
<mx:TileList dataProvider="{TileListdp}"
 itemRenderer="mx.controls.Button"/>
</s:Application>
```

Les grilles

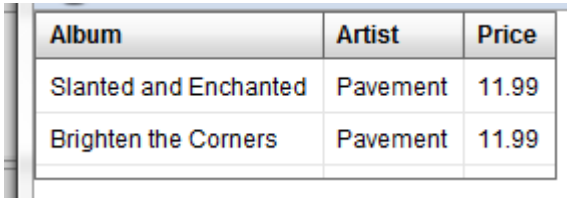
- DataGrid
  - Permet de
    - ranger des informations différentes selon plusieurs colonnes
    - d'avoir des intitulés de colonnes
    - de pouvoir cliquer dessus pour trier les contenus
    - de masquer des colonnes
    - de lier (binder) les colonnes pour les modifier automatiquement
    - de remplir les colonnes par différents fournisseurs : listes d'objets, tableaux, webservices, etc.

Les grilles

- DataGrid (suite)
  - Le fournisseur de données consiste en une liste d'objets appelés *éléments*
    - Chaque élément correspond à une ligne du DataGrid
    - Chaque colonne de la grille correspond typiquement à une propriété de chaque élément de ligne
  - On peut utiliser plusieurs types de fournisseurs de ce type d'élément
  - Dans le cas le plus simple :
    - <s:ArrayList> pour la structure et <fx:Object> pour les entrées
    - Chaque objet définit une ligne du contrôle DataGrid

Exemple : datagrid1.mxml

```
<s:Application …>
<s:DataGrid>
    <s:dataProvider>
      <s:ArrayList>
                <fx:Object>
                        <fx:Artist>Pavement</fx:Artist>
                        <fx:Price>11.99</fx:Price>
                        <fx:Album>Slanted and Enchanted</fx:Album>
                </fx:Object>
                <fx:Object>
                        <fx:Price>11.99</fx:Price>
                        <fx:Artist>Pavement</fx:Artist>
                        <fx:Album>Brighten the Corners </fx:Album>
                </fx:Object>
      </s:ArrayList>
</s:dataProvider>
</s:DataGrid>
</s:Application>
```



Album	Artist	Price
Slanted and Enchanted	Pavement	11.99
Brighten the Corners	Pavement	11.99

Les grilles

- Comment spécifier le fournisseur de données ?
  - On utilise la propriété dataProvider
    - Le type de dataProvider est Ilist
    - Cela vous permet d'utiliser un ArrayList (comme vu précédemment), un ArrayCollection, ou toute aute classe qui implémente IList
  - ArrayList est une implementation légère de IList qui est utile si vous ne faites pas de tri de ligne
  - Pour faire du tri,
    - utilisez une classe qui implémente l'interface ICollectionView, comme ListCollectionView ou une une sous-classe de celle-ci, comme ArrayCollection ou XMLListCollection

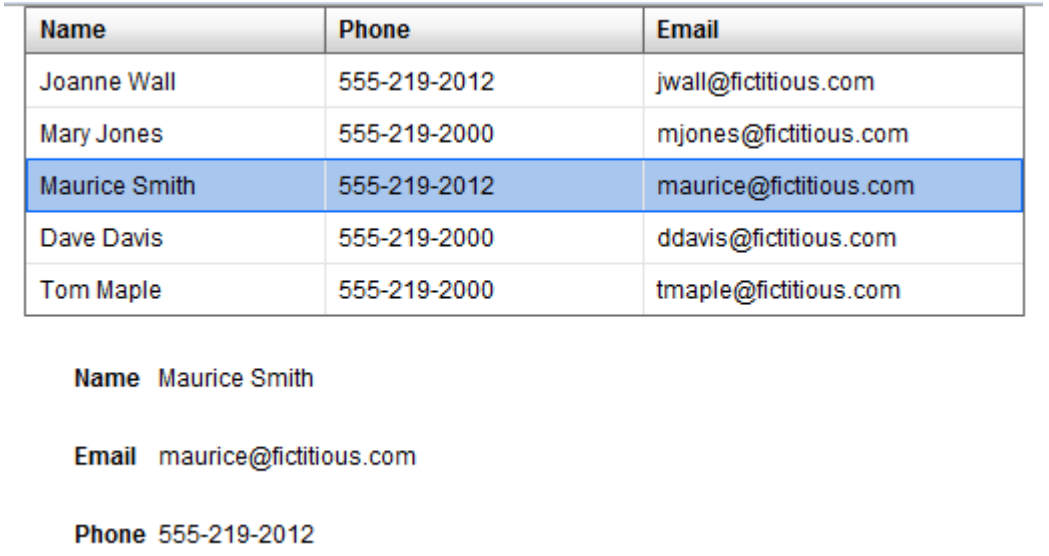
Les grilles

- Comment passer les données à DataGrid avant de les organiser en ArrayList, ArrayCollection…?
  - Flex vous permet de remplir un DataGrid de contrôle à partir d'une définition de variable ActionScript ou à partir d'un modèle de données Flex

- Cet exemple remplit un contrôle DataGrid en utilisant une variable : datagrid2.mxml

```
<s:Application …initialize="initData();">
<fx:Script>
<![CDATA[
    import mx.collections.*;
    private var dgArray:Array = [ {Artist:'Pavement',  Album:'Slanted and
    Enchanted', Price:11.99},
    {Artist:'Pavement', Album:'Brighten the Corners', Price:11.99}];
    [Bindable]
    public var initDG:ArrayCollection;
    // Initialise initDG à partir du tableau
    public function initData():void {
       initDG = new ArrayCollection(dgArray);
    }
]]>
</fx:Script>
//On lie la variable initDG et la propriété  <s:dataProvider>
<s:DataGrid id="myGrid"  width="350" height="200"
    dataProvider ="{initDG}"/>
</s:Application>
```

- Ici, on remplit le contrôle DataGrid à partir de données XML : datagrid3.mxml
  - On définit les données XML en utilisant la classe XMLList
  - On crée ensuite un objet XMLListCollection de la XMLList :
  - Quand on clique sur une ligne, les informations s'affichent en dessous de la grille



- Le code : datagrid3.mxml

```
<s:Application…>
<s:layout><s:VerticalLayout/></s:layout>
<fx:Declarations>
    <fx:XMLList id="employees">
      <employee>
                <name>Joanne Wall</name>
                <phone>555-219-2012</phone>
                <email>jwall@fictitious.com</email>
                <active>true</active>
      </employee>
      <employee>
                <name>Mary Jones</name>
                <phone>555-219-2000</phone>
                <email>mjones@fictitious.com</email>
                <active>true</active>
      </employee>…
    </fx:XMLList>
    <s:XMLListCollection id="employees2" source="{employees}"/>
</fx:Declarations>
```

- datagrid3.mxml (suite)

```
<s:DataGrid id="dg" width="500" dataProvider="{employees2}">
    <s:columns>
      <s:ArrayList>
                <s:GridColumn dataField="name" headerText="Name"/>
                <s:GridColumn dataField="phone" headerText="Phone"/>
                <s:GridColumn dataField="email" headerText="Email"/>
      </s:ArrayList>
    </s:columns>
</s:DataGrid>
<s:Form>
    <s:FormItem label="Name">
      <s:Label text="{dg.selectedItem.name}"/>
    </s:FormItem>
    <s:FormItem label="Email">
      <s:Label text="{dg.selectedItem.email}"/>
    </s:FormItem>
    <s:FormItem label="Phone">
      <s:Label text="{dg.selectedItem.phone}"/>
    </s:FormItem>
</s:Form>
</s:Application>
```

dataGrid

- Configurer les colonnes
  - Par défaut, le DataGrid crée une colonne pour chaque propriété dans le fournisseur
  - Les propriétés utiles pour manipuler ces colonnes sont :
    - <s:GridColumn> qui permet de définir l'objet colonne
    - <s:columns> qui permet d'afficher seulement les colonnes correspondant aux objets GridColumn
    - <s:typicalItem> qui permet de désigner un élément typique

- Exemple : datagrid4.mxml

```
<s:Application  …">
<s:DataGrid>
    <s:dataProvider>
      <s:ArrayCollection>
                <fx:Object Artist="Pavement" Price="11.99"
                Album="Slanted and Enchanted"/>
                <fx:Object Artist="Pavement"
                Album="Brighten the Corners" Price="11.99"/>
      </s:ArrayCollection>
    </s:dataProvider>
    <s:columns> //voir explications après
      <s:ArrayList>
                <s:GridColumn dataField="Album"/>
                <s:GridColumn dataField="Price"/>
      </s:ArrayList>
    </s:columns>
</s:DataGrid>
</s:Application>
```

- Intérêt de <s:columns> et s:GridColumn>
  - Dans cet exemple, on a défini des colonnes pour les propriétés Album et Price du fournisseur de données
  - Par conséquent, la DataGrid n'affiche pas une colonne pour la propriété Prix
  - On peut réordonner les colonnes en changeant l'ordre des objets GridColumn, comme ceci :

```
<s:columns>
    <s:ArrayList>
            <s:GridColumn dataField="Price"/>
            <s:GridColumn dataField="Album"/>
    </s:ArrayList>
</s:columns>
```

  - Ici, on indique que la colonne Price est la première colonne du DataGrid et que la colonne Album est la seconde

- Intérêt de <s:GridColumn>
  - On peut utiliser <s:GridColumn> pour mettre d'autres options
    - Ici, on utilise la propriété headerText pour mettre le nom de la colonne à une valeur différente de la valeur par défaut de Album,
    - et la propriété width pour expliciter la largeur d'une colonne :

```
<s:columns>
  <s:ArrayList>
    <s:GridColumn dataField="Album" width="200"/>
    <s:GridColumn dataField="Price" headerText="List
            Price"/>
  </s:ArrayList>
</s:columns>
```

- la propriété <s:typicalItem>
  - Le DataGrid utilise des propriétés pour définir un élément typique, et tout rendu d'élément associé à une colonne, pour calculer la largeur initiale de chaque colonne
  - Dans comme l'exemple suivant : datagrid5.mxml on utilise la propriété typicalItem pour préciser les caractéristiques l'élément de données colonne

Suite : datagrid5.mxml

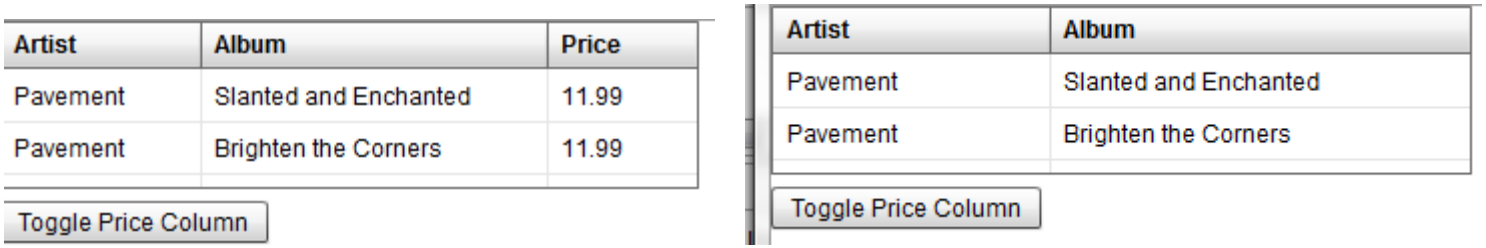
```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark">

    <s:DataGrid requestedRowCount="5">
      <s:typicalItem>
            <s:DataItem key="9999999" name="Typical name length"
                  price="1234.56" call="false"/>
      </s:typicalItem>
      <s:ArrayCollection id="items">
            <s:DataItem key="1000" name="Abrasive" price="100.11" call="false"/>
            <s:DataItem key="1001" name="Brush" price="110.01" call="true"/>
            <s:DataItem key="1002" name="Clamp" price="120.02" call="false"/>
            <s:DataItem key="1003" name="Drill" price="130.03" call="true"/>
            <s:DataItem key="1004" name="Epoxy" price="140.04" call="false"/>
            <s:DataItem key="1005" name="File" price="150.05" call="true"/>
      </s:ArrayCollection>
    </s:DataGrid>
</s:Application>
```

Trier les colonnes de dataGrid

- Par défaut, le contrôle DataGrid affiche des lignes dans l'ordre spécifié dans les articles de données passé à sa propriété dataProvider
- Le contrôle vous permet de trier les données basées sur la valeur de la cellule d'une colonne unique
- Pour trier par ordre croissant d'une colonne, sélectionnez l'en-tête de colonne
  - Sélectionnez-le à nouveau pour trier dans l'ordre décroissant
- Pour désactiver le tri pour le contrôle
  - Mettez la propriété DataGrid.sortableColumns à false
- Pour désactiver le tri pour une colonne
  - Mettez la propriété GridColumn.sortable à false
- Vous pouvez créer un tri personnalisé en mettant en œuvre une sorte fonction de comparaison

Cacher et visualiser les colonnes

- Si vous affichez une colonne à certains moments, mais pas à d'autres, vous pouvez définir la propriété visible de GridColumn pour afficher ou masquer la colonne
- L'exemple suivant vous permet de masquer ou d'afficher les prix de l'album en cliquant sur un bouton:



- Datagrid6.mxml

```
<s:Application …>
<s:layout> <s:VerticalLayout/> </s:layout>
<s:DataGrid id="myDG" width="350">
    <s:ArrayCollection>
      <fx:Object Artist="Pavement" Price="11.99" Album="Slanted and Enchanted"/>
      <fx:Object Artist="Pavement" Album="Brighten the Corners" Price="11.99"/>
    </s:ArrayCollection>
    <s:columns>
      <s:ArrayList>
                <s:GridColumn dataField="Artist"/>
                <s:GridColumn dataField="Album"/>
                <s:GridColumn id="price" dataField="Price" visible="false"/>
      </s:ArrayList>
    </s:columns>
</s:DataGrid>

<!-- The column id property specifies the column to show.-->
<s:Button label="Toggle Price Column" click="price.visible = !price.visible;" />
</s:Application>
```

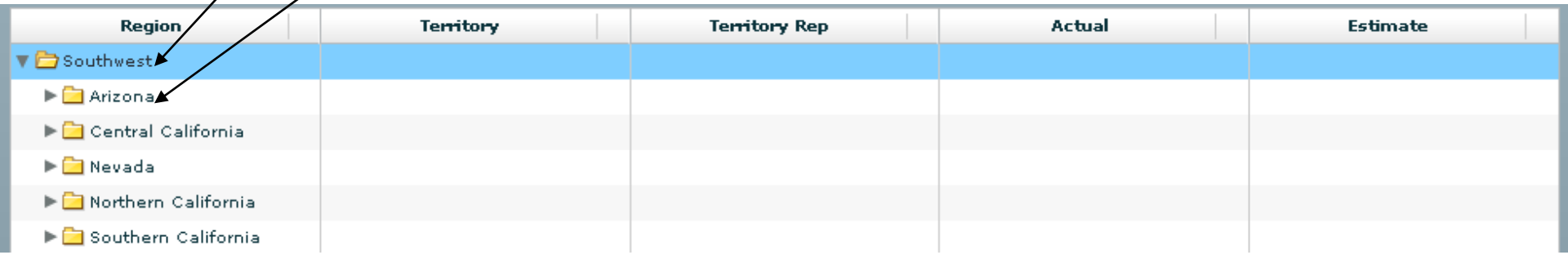
Les contrôles avancés

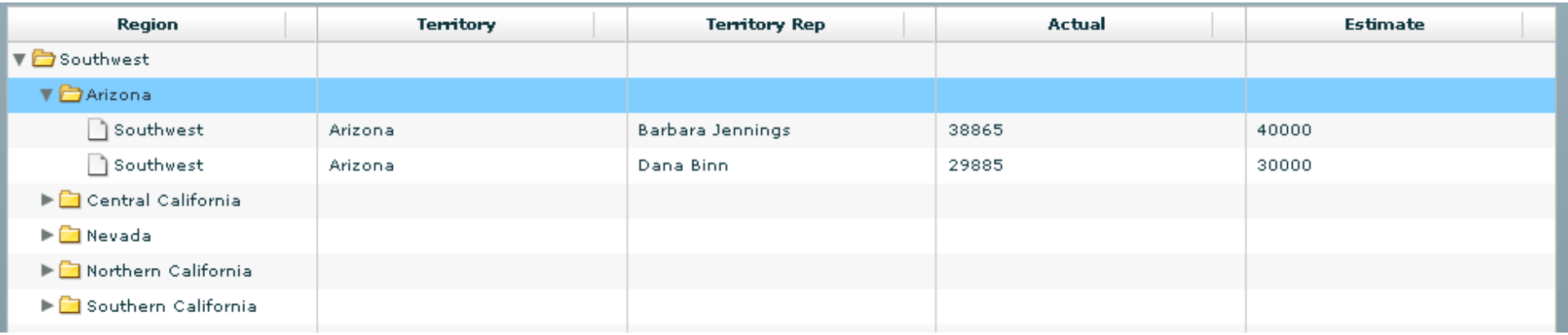
- L'AdvancedDataGrid : advanceddg.mxml
  - Grâce à son provider <mx:GroupingCollection>, ce composant prend une donnée à plat et essaie de l'organiser en hiérarchie pour pouvoir l'afficher
  - Pour grouper la data, on doit spécifier un ou plusieurs champ pour pouvoir créer la hiérarchie

- Exemple
  - Voici notre donnée "flat" (à plat) : SimpleFlatData.as

```
[Bindable]
private var dpFlat:ArrayCollection = new ArrayCollection([
{Region:"Southwest", Territory:"Arizona",
   Territory_Rep:"Barbara Jennings", Actual:38865, Estimate:40000},
{Region:"Southwest", Territory:"Arizona",
   Territory_Rep:"Dana Binn", Actual:29885, Estimate:30000},
{Region:"Southwest", Territory:"Central California",
   Territory_Rep:"Joe Smith", Actual:29134, Estimate:30000},
{Region:"Southwest", Territory:"Nevada",
   Territory_Rep:"Bethany Pittman", Actual:52888, Estimate:45000},
{Region:"Southwest", Territory:"Northern California",
   Territory_Rep:"Lauren Ipsum", Actual:38805, Estimate:40000},
{Region:"Southwest", Territory:"Northern California",
   Territory_Rep:"T.R. Smith", Actual:55498, Estimate:40000},
{Region:"Southwest", Territory:"Southern California",
   Territory_Rep:"Alice Treu", Actual:44985, Estimate:45000},
{Region:"Southwest", Territory:"Southern California",
   Territory_Rep:"Jane Grove", Actual:44913, Estimate:45000}
]);
```

- On spécifie **deux champs qui seront utilisés pour grouper** la donnée : Region et Territory
- La première colonne du composant AdvancedDataGrid est associée à ces deux champs
  - donc l'arbre de navigation interne utilise le champ Region pour définir les labels de chaque noeud enfant (un territoire) :

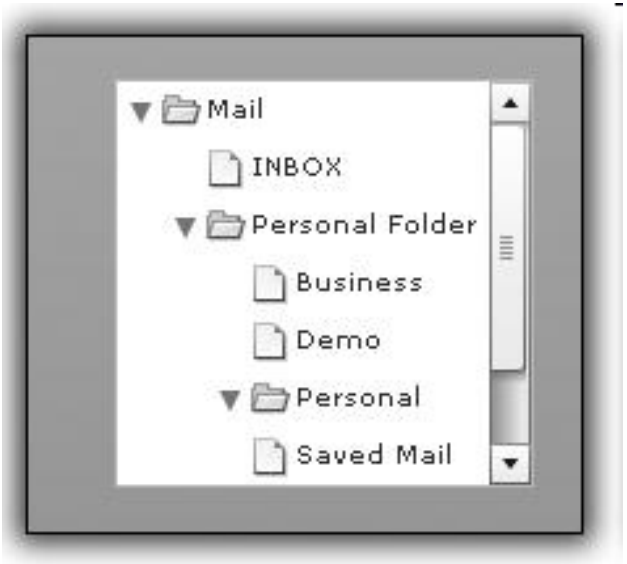




```
…import mx.collections.ArrayCollection;
        include "SimpleFlatData.as"
    ]]>
    </mx:Script>
    <mx:AdvancedDataGrid id="myADG" width="100%" height="100%"
      initialize="gc.refresh();">
   //hiérarchie de groupement suivant Region et Territory (1ère colonne)
    <mx:dataProvider>
        <mx:GroupingCollection id="gc" source="{dpFlat}">
            <mx:grouping>
                <mx:Grouping>
                    <mx:GroupingField name="Region"/>
                    <mx:GroupingField name="Territory"/>
                </mx:Grouping>
            </mx:grouping>
        </mx:GroupingCollection>
    </mx:dataProvider>
    //Rangement des autres éléments dans les colonnes suivantes
    <mx:columns>
        <mx:AdvancedDataGridColumn dataField="Region"/>
        <mx:AdvancedDataGridColumn dataField="Territory"/>
        <mx:AdvancedDataGridColumn dataField="Territory_Rep"
            headerText="Territory Rep"/>
        <mx:AdvancedDataGridColumn dataField="Actual"/>
        <mx:AdvancedDataGridColumn dataField="Estimate"/>
        </mx:columns>
    </mx:AdvancedDataGrid>
</mx:Application>
```

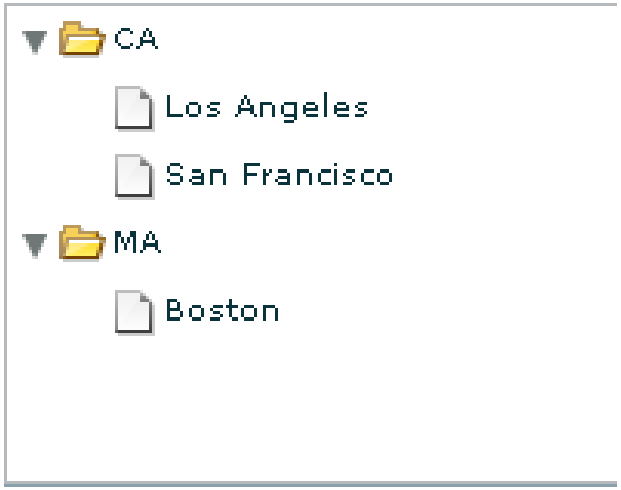
Les arbres

- Tree
  - Permet de présenter des données hiérarchisées
  - Les providers sont de type XML et XMLList



Les contrôles avancés

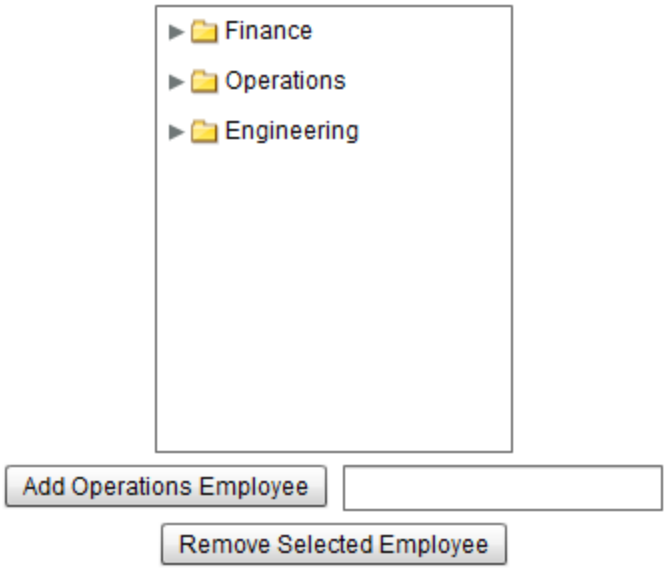
- Exemple de remplissage avec un XMLList : tree.mxml




```
<mx:Tree labelField="@label" width="200">
    <mx:XMLListCollection>
        <mx:XMLList>
            <state label="CA">
                <city label="Los Angeles"/>
                <city label="San Francisco"/>
            </state>
            <state label="MA">
                <city label="Boston"/>
            </state>
        </mx:XMLList>
    </mx:XMLListCollection>
</mx:Tree>
```

Autre exemple : tree2.mxml

- Le provider est un arbre XML
  - Voici la liste des données




```
import mx.collections.XMLListCollection;
[Bindable]
private var company:XML =
<list>
    <department title="Finance"
            code="200">
            <employee name="John H"/>
            <employee name="Sam K"/>
    </department>
    <department title="Operations" code="400">
            <employee name="Bill C"/>
            <employee name="Jill W"/>
    </department>
    <department title="Engineering" code="300">
            <employee name="Erin M"/>
            <employee name="Ann B"/>
    </department>
</list>;
```

```
[Bindable]
private var companyData:XMLListCollection = new XMLListCollection(company.department);

    private function treeLabel(item:Object):String{
          var node:XML = XML(item);
           if( node.localName() == "department" ) return node.@title;
          else return node.@name;
     }
     private function addEmployee():void{
        var newNode:XML = <employee/>;
        newNode.@name = empName.text;
        var dept:XMLList =company.department.(@title == "Operations");
        if( dept.length() > 0 ) {
            dept[0].appendChild(newNode);
            empName.text = "";
         }
    }
    private function removeEmployee():void{
          var node:XML = XML(tree.selectedItem);
           if( node == null ) return;
           if( node.localName() != "employee" ) return;
           var children:XMLList = XMLList(node.parent()).children();
           for(var i:Number=0; i < children.length(); i++) {
                        if( children[i].@name == node.@name ) {
                                    delete children[i];}
                        }
           }
    }
  ]]>
</mx:Script>
```

La variable XML

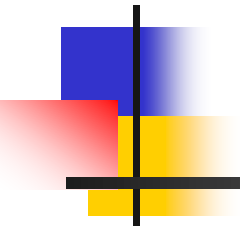
```
<mx:Tree id="tree" top="72" left="50" dataProvider="{companyData}"
        labelFunction="treeLabel" height="224" width="179"/>
<mx:HBox>
        <mx:Button label="Add Operations Employee" click="addEmployee()"/>
         <mx:TextInput id="empName"/>
</mx:HBox>
```

TD1

- Énoncé
  - Reprenez l'exemple précédent et contrôlez l'insertion :
    - Pas d'insertion d'une opération vide : contrôle sur la valeur du champ texte
    - Ajout que d'une opération qui n'existe pas
    - On ne peut pas détruire une opération inexistante.
  - Faites la même chose sur les autres rubriques : Finance…

Aller vers une interface riche

Créer ses propres composants



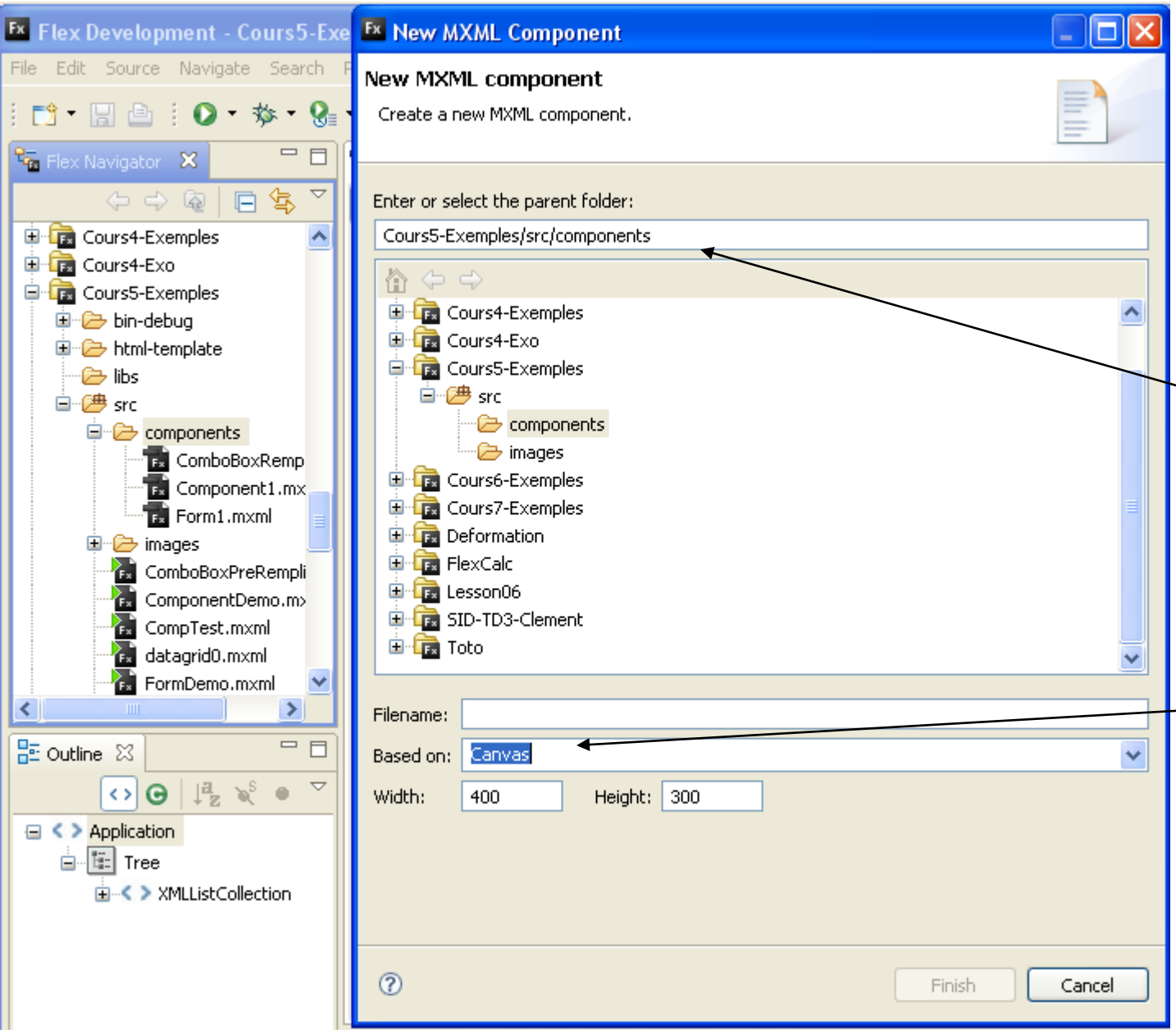
Créer ses propres composants

- Principe
  - Flex permet de créer ses propres composants à partir de composants déjà existants
    - Un composant Flex ne commence pas par <mx:Applications>

    mais par :
    - <mx:Box>, <mx:Form>, <mx:Canvas>, etc.
  - Ensuite, vous pouvez spécialiser un composant existant pour le rendre intégrable directement dans beaucoup d'applications

Créer ses propres composants

- La création de composants Flex est très facile
  - Créez un répertoire components (s'il n'a pas été créé lors de la création du projet) :
    - Flex puis **New**, **Folder** et appelez le "components"
  - Un composant se crée de cette manière
    - Cliquer sur components
    - Ensuite sur Flex (Interface générale) >>File >>New >>MXML Component



important

Sélectionnez le composant de votre choix

Créer ses propres composants

- Exemple : le composant de base est <mx:Form>
  - **Loginform.mxml**

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Form xmlns:mx="http://www.adobe.com/2006/mxml"
         width="268" height="124">
    <mx:FormItem label="Login :">
        <mx:TextInput id="login"
        displayAsPassword="false" editable="true"/>
    </mx:FormItem>
    <mx:FormItem label="Password :">
        <mx:TextInput id="password"
        displayAsPassword="true" editable="true"/>
    </mx:FormItem>
    <mx:FormItem>
        <mx:Button label="S'enregistrer"/>
    </mx:FormItem>
</mx:Form>
```

La première balise n'est pas <mx:Application>

  - On va pouvoir ensuite en une seule ligne ajouter ce nouveau composant dans nos applications Flex

Créer ses propres composants

- Utilisation du composant de base
  - Maintenant que le composant personnalisé est créé et se trouve dans le dossier components, il faut lui créer un namespace pour pouvoir l'utiliser
  - Namespace
    - Tous les composants fournis par Flex s'appellent dans le namespace mx
    - Or, ce nouveau composant n'est pas fourni par Flex
    - Il faut lui créer un namespace pour ne pas risquer de conflit
    - Voici la syntaxe :

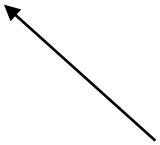
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application  xmlns:mx="http://www.adobe.com/2006/
   mxml"  xmlns:mcmp="components.*" layout="absolute">
```

    - Dorénavant, lorsque on ouvre une balise MXML en indiquant le namespace mcmp, Flex Builder proposera en complétion automatique tous les composants auxquels fait référence mcmp

Créer ses propres composants

- Utilisation du composant de base
  - Ajout du composant
    - se fait par simple ajout de balise dans le code MXML
  - Exemple : utiliseForm.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
   xmlns:mx="http://www.adobe.com/2006/mxml"xmlns:mcomp
   ="components.* "layout="absolute">
   <mx:Panel title="login">
            <mcomp:Loginform />
   </mx:Panel>
</mx:Application>
```



Composant créé précédemment

Utilisation du composant de base

- Le composant de base peut contenir du code ActionScript
  - La balise <mx:Script> permet bien sûr de définir des propriétés ou des méthodes dans les composants personnels
  - Position de la balise <mx:Script>
    - La balise <mx:Script> doit être un enfant immédiat de la racine du fichier MXML
    - Toute déclaration publique de fonctions ou de variables devient alors une méthode ou une propriété du composant créé
    - Par exemple, voici un bout de code dans lequel maPropriete et get_propriete() sont une propriété et une méthode du composant monTextArea :

Utilisation du composant de base

- Exemple : monTextArea.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TextArea xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
    <![CDATA[
    // Ici la variable qui devient de fait propriété du composant :
    public var maPropriete:Number;
    // Cette fonction est une méthode du composant
    public function get_propriete(){
    return "Le nombre indiqué dans la propriété maPropriete est :
        " +maPropriete;
    }
    ]]>
    </mx:Script>
</mx:TextArea>
```

Première balise

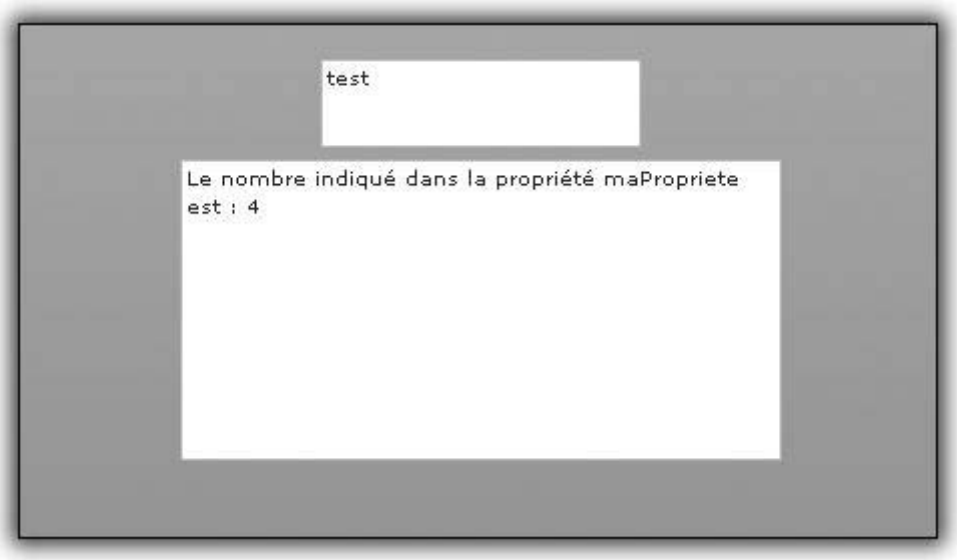
Utilisation du composant de base

- Voici maintenant une application qui l'utilise :
  changePropriete.mxml

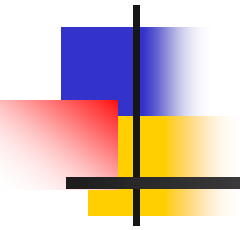
```
<?xml version="1.0"?>
<mx:Application
   xmlns:mx="http://www.adobe.com/2006/mxml"xmlns:mcmp="c
   omponents.*">
   <mcmp:monTextArea id="mta" maPropriete="4"
            change="ta1.text = mta.get_propriete();"/>
   <mx:TextArea id="ta1" width="300" height="150" />
</mx:Application>
```

- À l'exécution, on va voir que lorsque adviendra un changement dans le TextArea mta, le TextArea ta1 se remplira de la valeur : "Le nombre indiqué dans la propriété maPropriete est : 4".

Utilisation du composant de base



Exemple avec les TextArea



TD2

- Énoncé
  - Créer un objet à vous, en y ajoutant du code ActionScript
    - Attention, sous Spark, utiliser les groupes qu'il vous propose
    - Par exemple, si vous voulez créer un bouton, vous créer un component Group, puis, à l'intérieur, vous ajoutez un bouton
  - Donner une application qui l'utilise

Interface Maître-Détail

Interface Maître-Détail

- Qu'est-ce que c'est ?
  - Dans une interface maître-détail, l'utilisateur peut sélectionner des objets à partir d'une liste d'objets et visualiser les objets sélectionnés
    - L'interface maître affiche les objets sélectionnés
    - L'interface détail met en œuvre un visualiseur de l'objet sélectionné
  - Chaque fois que l'utilisateur modifie la sélection dans l'interface maître, l'interface détail est mise à jour pour afficher la nouvelle sélection
  - Si aucun objet n'est sélectionné, le détail se désactive lui-même
  - Si plusieurs objets sont sélectionnés, le détail est désactivé ou s'applique à tous les objets sélectionnés

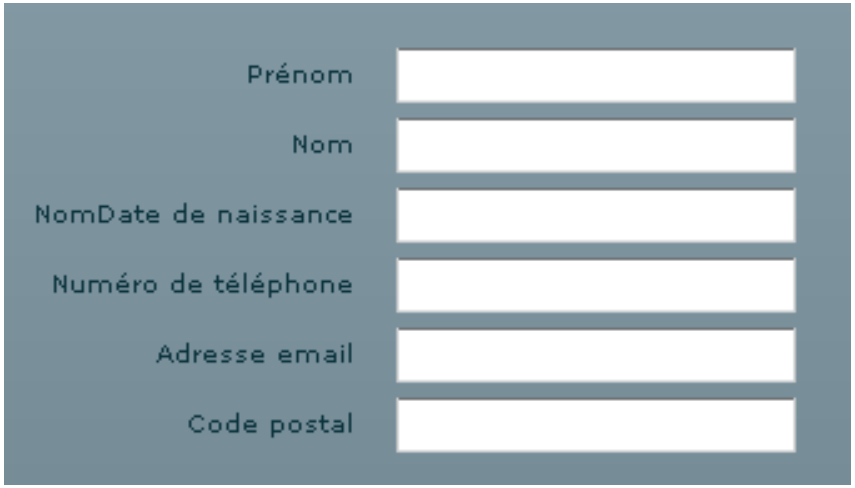
Interface Maître-Détail

- Exemple d'une interface maître-détail

Maître

Id	Nom	Prénom
1	Edouard	Edouard
2	Baron	Vianne
3	Bernal	Jessy

Détail

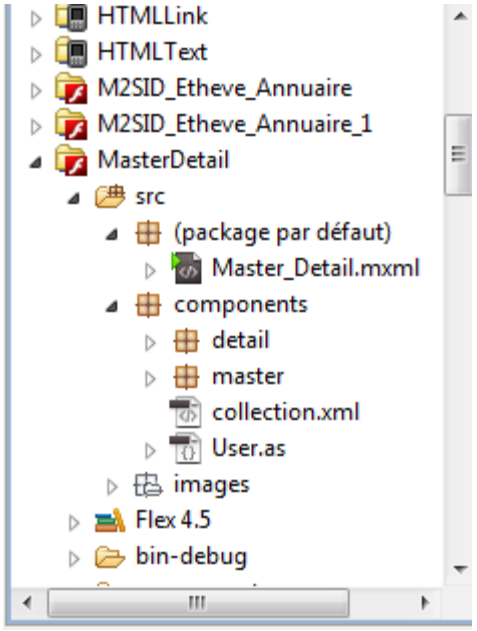


Interface Maître-Détail

- Mise en pratique : une interface maître-détail
  - Mise en place du projet
    - Créer un nouveau projet appelé MasterDetail
    - Créer aussi tout de suite le dossier components qui servira à stocker les différents modules
  - Le module components contiendra :
    - la collection
      - un fichier XML
    - deux modules
      - le module maître qui contiendra l'interface maître
      - et le module détail qui contiendra l'interface détail

Interface Maître-Détail

- Mise en pratique : une interface maître-détail
  - Voici la nouvelle arborescence du projet



Interface Maître-Détail

Le module maître

- La collection
  - le fichier XML (ici collection.xml) sera utilisé par les composants master et detail
  - Ce fichier, vu comme un modèle, comporte pour chaque enregistrement user, plusieurs champs :
    - id pour l'identifiant,
    - lastname et firstname (pour le nom et le prénom),
    - birthday (pour la date de naissance),
    - ainsi que phone,
    - email et zipcode pour respectivement le numéro de téléphone, l'adresse e-mail et le code postal de chaque personne enregistrée

- collection.xml

```
<?xml version="1.0" encoding="utf-8"?>
<collec>
  <user>
      <id>1</id>
      <firstname>Edouard</firstname>
      <lastname>Ruiz</lastname>
      <birthday>10/05/1984</birthday>
      <phone>0654321098</phone>
      <email>edouard@yahoo.com
      </email>
      <zipcode>94130</zipcode>
  </user>
  <user>
      <id>2</id>
      <firstname>Vianney</firstname>
      <lastname>Baron</lastname>
      <birthday>14/07/1985</birthday>
       <phone>0658982345</phone>
<email>vianney@yahoo.com
    </email>
    <zipcode>75017</zipcode>
  </user>
<user>
    <id>3</id>
    <firstname>Jessy</firstname>
    <lastname>Bernal</lastname>
    <birthday>23/01/1985</birthday>
    <phone>0698235455</phone>
    <email>Jessy@yahoo.com
    </email>
    <zipcode>94800</zipcode>
  </user>
</collec>
```

- Le module maître
  - Pour la création du module maître, nous allons enrichir le composant Box avec un DataGrid que nous allons remplir avec les données de la collection

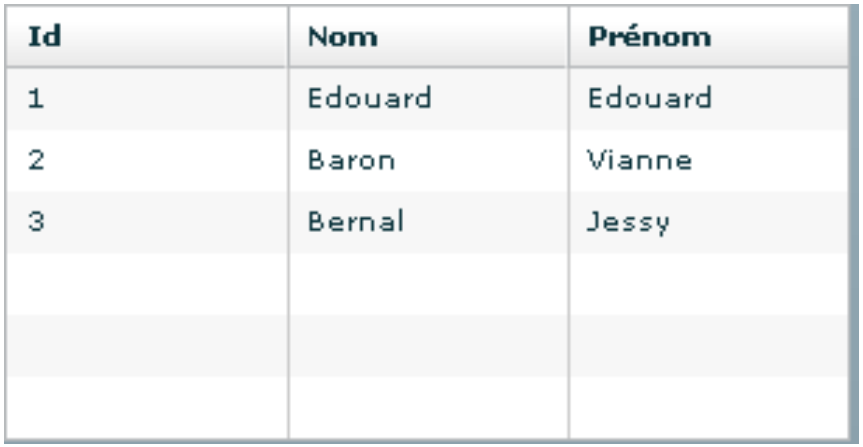
```
<mx:Box xmlns:mx="http://www.adobe.com/2006/mxml"
width="400"  height="300">
    <mx:ArrayCollection id="arrayUser"/>
    <mx:DataGrid id="masterList" allowMultipleSelection="false" >
        <mx:columns>
            <mx:Array>
                <mx:DataGridColumn headerText="Id"
                    dataField="id"/>
                <mx:DataGridColumn headerText="Nom"
                    dataField="lastname"/>
                <mx:DataGridColumn headerText="Prénom"
                    dataField="firstname"/>
            </mx:Array>
        </mx:columns>
    </mx:DataGrid>
</mx:Box>
```

- Version plus complète : La Box contient le lien du modèle à la dataGrid

```
<mx:Box xmlns:mx="http://www.adobe.com/2006/mxml" width="400" height="300">
    //On indique où se trouve la collection
    <mx:Model id="col" source="..\collection.xml"/>
    //On lie les données de ce modèle : col à arrayUser
    <mx:Binding source="col.user" destination="arrayUser.source"/>
     //On lie les données de arrayUser à la DataGrid
    <mx:Binding source="arrayUser" destination="masterList.dataProvider"/>


     <mx:ArrayCollection id="arrayUser"/>
        <mx:DataGrid id="masterList" allowMultipleSelection="false" >
            <mx:columns>
                <mx:Array>
                    <mx:DataGridColumn headerText="Id" dataField="id"/>
                    <mx:DataGridColumn headerText="Nom" dataField="lastname"/>
                    <mx:DataGridColumn headerText="Prénom" dataField="firstname"/>
                </mx:Array>
            </mx:columns>
        </mx:DataGrid>
</mx:Box>
```

- En faisant appel à ce nouveau module dans le fichier MasterDetail.mxml, nous obtenons le résultat escompté :

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute"  xmlns:master="components.master.*">
   <master:master />
</mx:Application>
```



Id	Nom	Prénom
1	Edouard	Edouard
2	Baron	Vianne
3	Bernal	Jessy

- Le module détail : detail.mxml
  - Comme pour le module maître, on part du composant <mx:Box> puis on y ajoute les éléments dont nous avons besoin
  - L'affichage des informations se fera à l'aide d'un formulaire

```
<mx:Form>
   <mx:FormItem label="Prénom">
            <mx:TextInput id="firstname"/>
   </mx:FormItem>
   <mx:FormItem label="Nom">
            <mx:TextInput id="lastname" text=""/>
   </mx:FormItem>
            <mx:FormItem label="NomDate de naissance">
            <mx:TextInput id="birthday"/>
   </mx:FormItem>
   <mx:FormItem label="Numéro de téléphone">
            <mx:TextInput id="phone"/>
   </mx:FormItem>
   <mx:FormItem label="Adresse email">
            <mx:TextInput id="email"/>
   </mx:FormItem>
   <mx:FormItem label="Code postal">
            <mx:TextInput id="zipcode"/>
   </mx:FormItem>
</mx:Form>
```

Le module détail (suite)

- Pour pouvoir récupérer les informations depuis l'interface maître, on crée une classe User avec des méthodes de récupération de ses variables d'instance
- Le code ActionScript ajouté au module detail est :

```
<mx:Script>
<![CDATA[
import components.User;
public function applyChange(user:User):void
{
    lastname.text = user.getLastName();
    firstname.text = user.getFirstName();
    birthday.text = user.getBirthday();
    email.text = user.getEmail();
    phone.text = user.getPhone();
    zipcode.text = user.getZipcode();
}
]]>
</mx:Script>
```

Interface Maître-Détail

Le module maître

- La classe User
  - Stockée dans le fichier User.as, lui-même placé dans le dossier components afin de la rendre facilement accessible aux deux modules, cette classe contient les sept propriétés qui vont contenir toutes les informations qui caractérisent un utilisateur, ainsi que les getters et setters
  - Sera instanciée lors d'un événement click sur un élément de la collection, et cet objet sera alors transmis à l'interface détail

```
package components{
    public class User{
        private var id:int;
        private var firstname : String;
        private var lastname : String;
        private var birthday : String;
        private var email : String;
        private var phone : String;
        private var zipcode : String;

        public function User(id:int,
            firstname:String,
            lastname:String,
            birthday:String,
            email:String,
            phone:String,
            zipcode:String):
         void{
           this.setid(id);
           this.setFirstname(firstname);
           this.setLastname(lastname);
           this.setBirthday(birthday);
           this.setEmail(email);
           this.setPhone(phone);
           this.setZipcode(zipcode);
         }
```

```
/*Getters et setters dont voici deux exemples */

        public function setEmail(value:String) : void
        {
                this.email=value;
        }
        public function getPhone() : String
        {
                return this.phone;
        }
        ...
}
```

Interface Maître-Détail

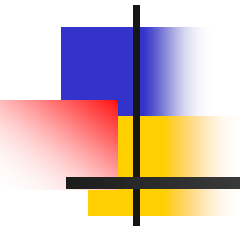
- L'assemblage des deux modules : MasterDetail.mxml

```
//on ajoute deux namespaces (un par module)
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"  xmlns:master="components.master.*"
    xmlns:detail="components.detail.*">


…
//ensuite, on insère la balise correspondante à chaque module
    dans le corps du fichier
    <master:master id="masterInterface" click="transferUser()"/>
    <detail:detail id="detailInterface"/>
</mx:VBox>
</mx:Application>
```

- Le code complet de MasterDetail.mxml

On utilise la fonction transferUser() à la place de getUser()

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
xmlns:master="components.master.*" xmlns:detail="components.detail.*">
    <mx:Script>
    <![CDATA[
       import components.User;
       import mx.controls.Alert;
       public function transferUser():void
       {
                Alert.show(masterInterface.masterList.selectedItem.zipcode);
                var user:User = new
                User(masterInterface.masterList.selectedItem.id,
                masterInterface.masterList.selectedItem.firstname,
                masterInterface.masterList.selectedItem.lastname,
                masterInterface.masterList.selectedItem.birthday,
                masterInterface.masterList.selectedItem.email,
                masterInterface.masterList.selectedItem.phone,
                masterInterface.masterList.selectedItem.zipcode);
                detailInterface.applyChange(user);}]]>
</mx:Script>
<mx:VBox horizontalAlign="center">
    <master:master id="masterInterface" click="transferUser()"/>
    <mx:HRule width="338" height="3"/>
    <detail:detail id="detailInterface"/>
</mx:VBox>
</mx:Application>
```



TD 3

- Énoncé
  - Inspirez vous de cet exemple pour présenter votre annuaire suivant ce type d’interface