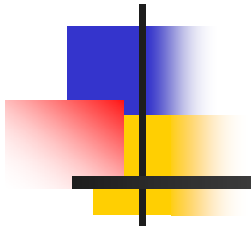
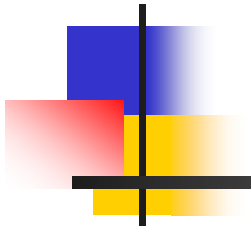
Flex

Aller vers une interface riche



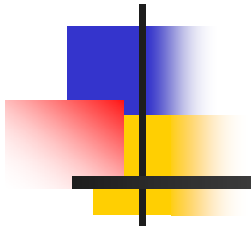
Aller vers une interface riche

- Introduction
  - Flex propose des composants tout faits (vus dans les chapitres précédents)
  - Il vous offre de plus des possibilités pour construire vos propres composants :
    - liste, tableau d'éléments "ordonnés" et de personnaliser vos applications
  - Pour cela, on va voir :
    - Les contrôles avancés
    - Comment créer ses propres composants



Les contrôles avancés

- Définition
  - Les contrôles avancés sont ceux qui sont dirigés par les données qu'ils sont chargés d'afficher
  - On trouve :
    - List,
    - HorizontalList,
    - ComboBox,
    - Tree
    - et DataGrid
  - Ces contrôles sont hérités de la classe ListBase

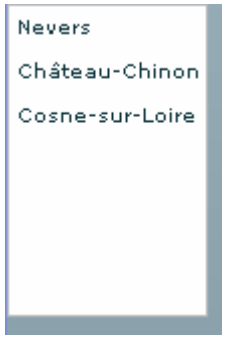


Les contrôles avancés

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

Les contrôles avancés

- Remplissage de List, HorizontalList
  - Se fait à l'aide de la propriété dataProvider entourant une liste de <mx:String> qui sera interprétée comme un tableau
  - Exemple : List.mxml

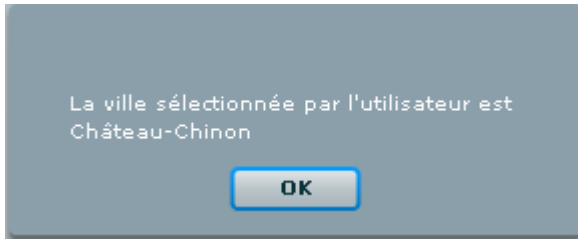



```
<mx:List id="ville_nievre">
   <mx:dataProvider>
         <mx:String>Nevers</mx:String>
         <mx:String>Château-Chinon</mx:String>
         <mx:String>Cosne-sur-Loire</mx:String>
   </mx:dataProvider>
</mx:List>
```

  - L'indexation des éléments se fait à partir de 0, ainsi "Nevers" aura l'index 0 et "Cosne-sur-Loire ", l'index 2

Remplissage de List, HorizontalList

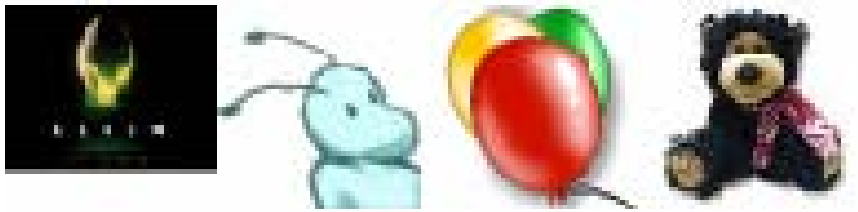
- La propriété selectedItem : selectedItem.mxml
  - Cette propriété permet de sélectionner un élément de la liste




```
<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>
```

Les contrôles avancés

- Remplir List avec des images : listImages.mxml
  - On crée un tableau d'images
  - Ce tableau sera le provider des images et sera vidé dans l'HorizontalList
  - Attention, pour que ces images soient visibles dans la liste, il faut préciser un "moteur de rendu", ou itemRenderer, spécifique au type de données à afficher, ici : itemRenderer="mx.controls.Image"
  - mx.controls.Image : permet d'importer des images de différents formats

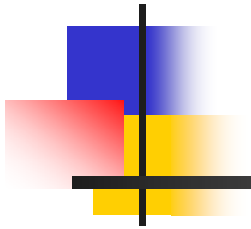


```
<mx:Script>
   <![CDATA[
            import mx.collections.*;
            import mx.controls.Image;
            private var imgArray:ArrayCollection;

            //On crée un tableau d'images
            private var cat:Array = ["images/Alien1.jpg",
            "images/Alien2.bmp", "images/Balloon.bmp",        "images/Bear.jpg"];

            public function initImgList(items:Array):void {
                        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)"
     //creationComplete déclenche un événement qui conduit à aller dans
     initImgList()
/>
```



List

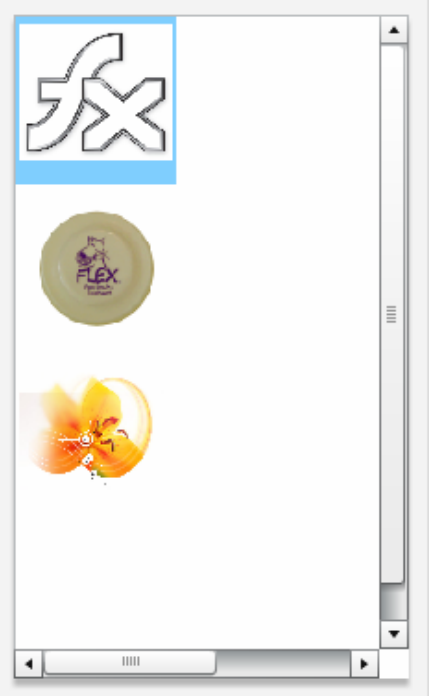
- TileList : gallerie.mxml
  - Le conteneur TileList agence ses enfants sur plusieurs lignes ou colonnes automatiquement
  - On voudrait utiliser cette TileList pour afficher une galerie d‘images
  - Le provider de TileList est ici une liste d’images récupérées d’un fichier XML : MyXML.xml
  - Pour ranger les images
    - On crée un Panel comprenant la TileList
  - Le moteur de rendu des images sera ici :
    - Une Vbox d’images cliquables
    - Quand on clique sur une image, on affiche l’image correspondante qui est plus grande
    - Cet 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 claiquables

Le canvas de l'image zoomée

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
   width="964" height="626" backgroundGradientColors="[#ffffff, #ffffff]">

<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

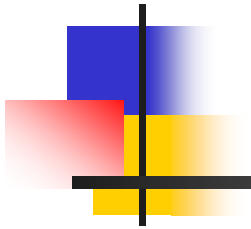
```
//Les fonctions ActionScript
<mx:Script>
    <![CDATA[
        import mx.events.CalendarLayoutChangeEvent;
        [Bindable]
        private var urlGdImg : String;

        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



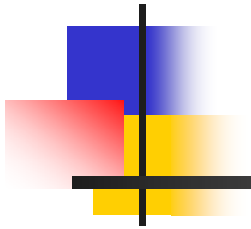
Les contrôles avancés

- Les DataGrid
  - Les données sont rangées sous forme de matrice
  - Ce type de contrôle permet de ranger des informations différentes selon plusieurs colonnes, d'avoir des intitulés de colonnes et de pouvoir cliquer dessus pour trier les contenus
  - Le remplissage de DataGrid se fait par l'intermédiaire de ArrayCollection permettant ainsi aux différents types de champs de former les différentes colonnes
  - ArrayCollection permet d'avoir un fils de type Array qui accepte comme élément Object
  - Les attributs de Object se rangent immédiatement dans les colonnes de DataGrid et les noms des attributs deviennent les entêtes des colonnes

- Exemple 1 : datagrid0.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
<mx:DataGrid>
    <mx:ArrayCollection>
      <mx:Array>
            <mx:Object city="Los Angeles" state="CA"
                  population="343943"/>
            <mx:Object city="New York" state="NY"
                  population="33444667"/>
            <mx:Object city="Chicago" state="IL"
                  population="343211245"/>
            <mx:Object city="Philadelphia" state="PA"
                  population="98976876"/>
      </mx:Array>
    </mx:ArrayCollection>
    </mx:DataGrid>
</mx:Application>
```

city	population	state
Los Angeles	343943	CA
New York	33444667	NY
Chicago	343211245	IL
Philadelphia	98976876	PA



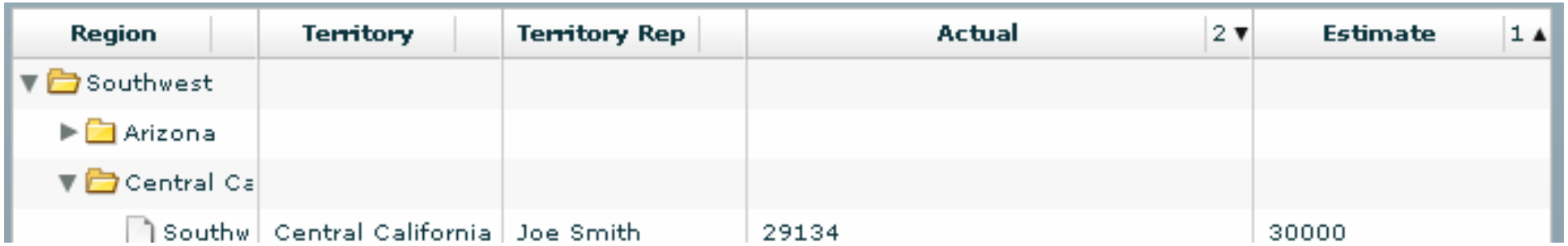
Les contrôles avancés

- L'AdvancedDataGrid
  - s'utilise comme un DataGrid
  - La différence se voit essentiellement à l'affichage, notamment au niveau du header avec une zone spécifique pour le tri
  - L'exemple advanceddg.mxml montre deux parties :
    - La description des colonnes
    - Et le provider

Advanced DataGrid

- La description des colonnes

```
<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>
```



Rangé dans **SimpleFlatData.as**

- Le provider

```
<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>
```

Les données sont dans les champs « Region » et « Territory »

– Le fichier : **SimpleFlatData.as**

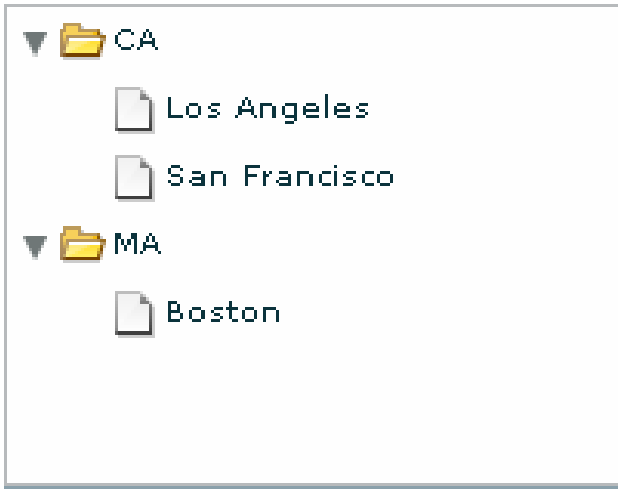
```
[Bindable]
private var dpFlat:ArrayCollection = new ArrayCollection([
{Region:"Southwest", Territory:"Arizona",
Territory_Rep:"Barbara Jennings", Actual:38865, Estimate:40000},
{Region:"Southwest", Territory:"Arizona",
Territory_Rep:"Dana Binn", Actual:29885, Estimate:30000},
{Region:"Southwest", 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}
]);
```

Les contrôles avancés

- Les Tree
  - Encore un type de liste différent qui permet d'afficher des données hiérarchisées
  - Les Tree permettent par exemple de présenter une arborescence de système de fichiers comme le fait l'explorateur Windows
  - Bien sûr, du fait de la hiérarchisation des données, le remplissage d'un Tree ne se fait pas à partir de simples Array ou ArrayCollection, mais nécessite l'utilisation des XML, et des 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>
```

Les contrôles avancés

- Autre exemple : tree2.mxml
  - Le provider est un arbre XML

```
<mx:Tree id="tree" top="72" left="50"
   dataProvider="{companyData}" labelFunction="treeLabel"
   height="224" width="179"/>
```

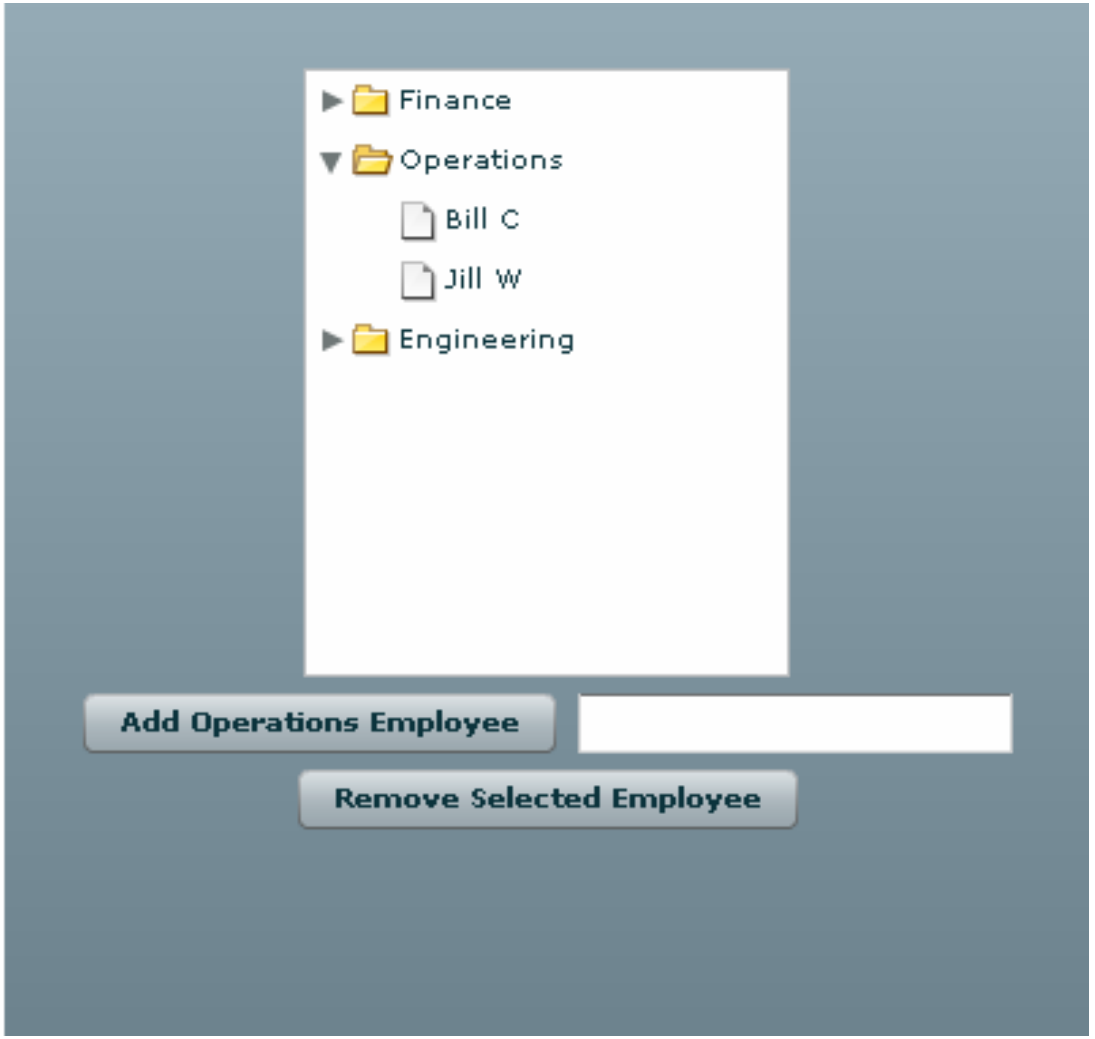
  - companyData correspond à la liste des départements de l'arbre XML présenté après

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

Les contrôles avancés

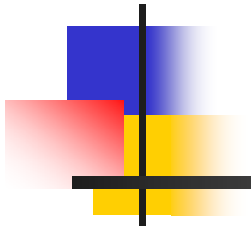
```
[Bindable]
        private var company:XML =
          <list>
           <department title="Finance" code="200">
              <employee name="John H"/>
              <employee name="Sam K"/>
           </department>
           <department title="Operations" code="400">
              <employee name="Bill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

Les contrôles avancés



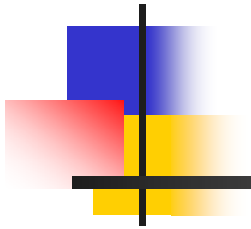
Aller vers une interface riche

Créer ses propres composants



Créer ses propres composants

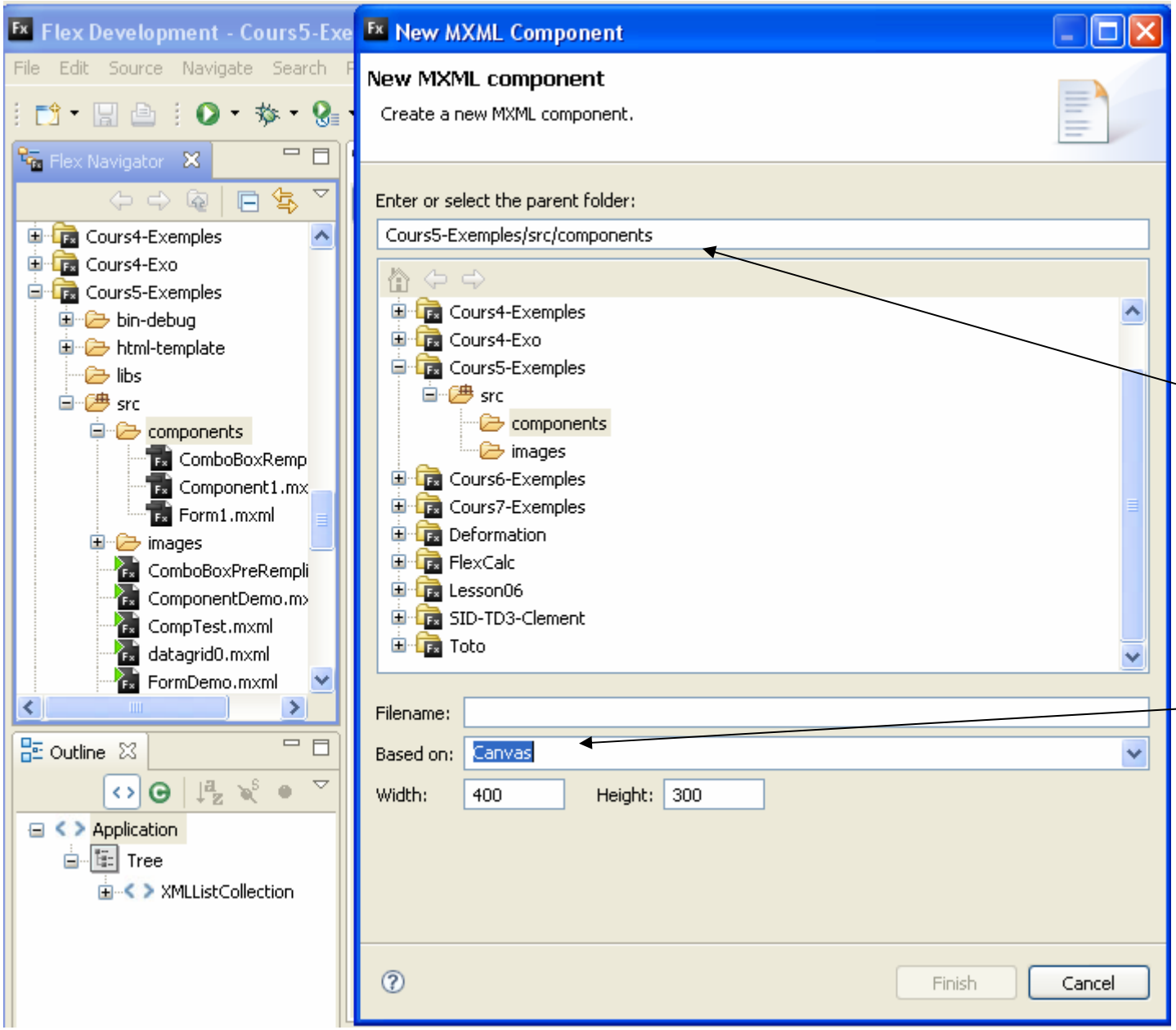
- Composants personnels
  - Flex vous permet de créer vos propres composants à partir de composants déjà existants
    - Un composant Flex ne commence pas par <mx:Applications>, mais par :
    - <mx:Box>, <mx:Form>, <mx:Canvas>, etc.
  - Ensuite, vous pouvez spécialiser un composant existant pour le 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">
```

Créer ses propres composants

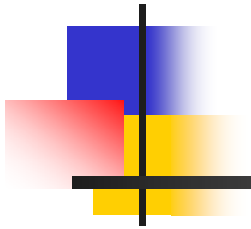
- Utilisation du composant de base
  - Dorénavant, lorsque on ouvre une balise MXML en indiquant le namespace mcmp, Flex Builder proposera en complétion automatique tous les composants auxquels fait référence mcmp

Exemple de complétion automatique pour le nouveau namespace

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:Applicationxmlns:mx="http://www.adobe.com/2006/mx
   ml"xmlns:mcomp="components.* "layout="absolute">
   <mx:Panel title="login">
            <mcomp:Loginform />
   </mx:Panel>
</mx:Application>
```



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

Première balise

```
<?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>
```

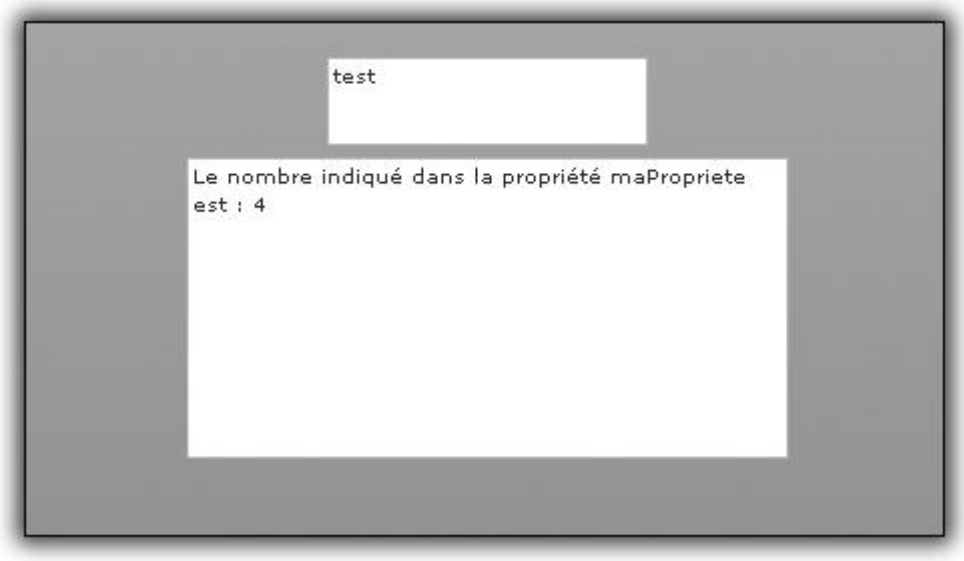
Utilisation du composant de base

- On peut utiliser la méthode ainsi définie : 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

Créer ses propres composants

- Les composants peuvent contenir du graphique
  - Exemple : RedButton.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Button xmlns:mx="http://www.adobe.com/2006/mxml">
</mx:Button>
```

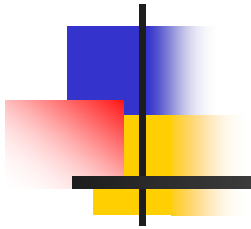
  - Ce bouton peut être utilisé dans une application comme celle-ci : utiliseButton.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   xmlns:comp="components.*" >
    <comp:RedButton label="Not so Red"/>
</mx:Application>
```

Créer ses propres composants

- Exemple : RedButton.mxml
  - Voici une autre version de RedButton en ActionScript avec un bouton rouge sur lequel on peut cliquer


```
<?xml version="1.0" encoding="utf-8"?>
<mx:Button xmlns:mx="http://www.adobe.com/2006/mxml"
   fillColors="['red', 'blue']" toolTip="A Red Button" click="clicked()">
   <mx:Script>
   <![CDATA[
            private function clicked():void {
                     label = "Quite Red";
                     setStyle("fillColors", ['red', 'red']);
            }
   ]]>
</mx:Script>
</mx:Button>
```



Mise en pratique :

créer une Interface Maître-Détail

- Qu'est-ce qu'une interface maître-détail ?
  - Dans une interface maître-détail, l'utilisateur peut sélectionner des objets à partir d'une liste d'objets et de 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 (s'il est modifiable)
  - Si plusieurs objets sont sélectionnés (en supposant que la sélection multiple est autorisée), le détail est désactivé ou s'applique à tous les objets sélectionnés
  - Généralement, cette interface permet également aux utilisateurs d'ajouter et de supprimer des objets de la collection

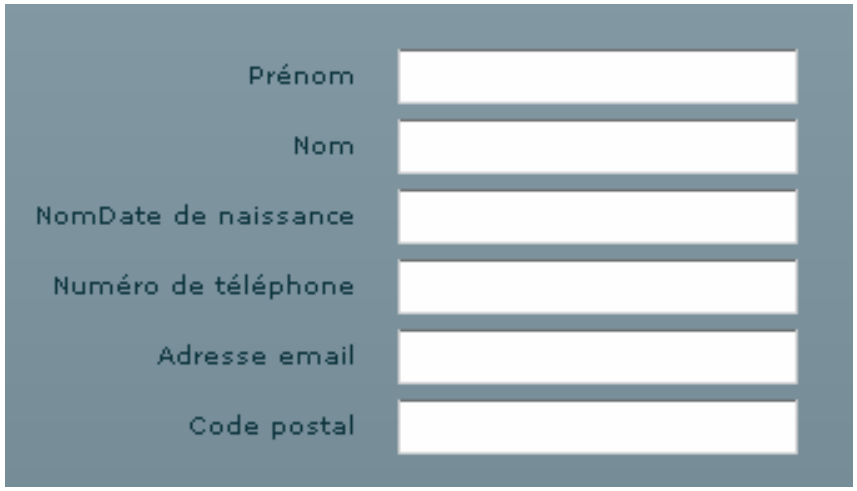
Interface Maître-Détail

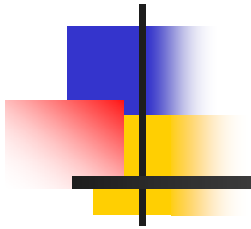
- Exemple d'une interface maître-détail

Maître

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

Détail



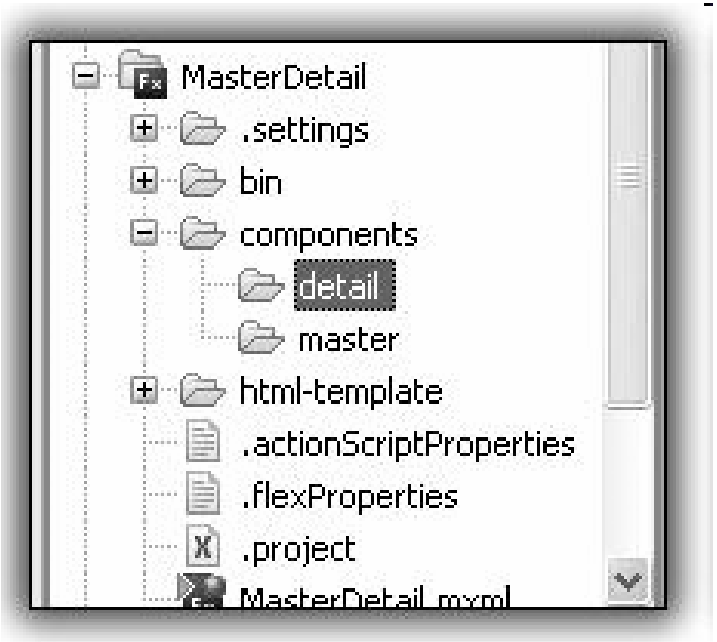


Interface Maître-Détail

- 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
  - Découpage du projet
    - La collection
      - Sera un fichier XML
    - Deux modules principaux
      - un module maître qui contiendra l'interface maître
      - et un module détail qui contiendra l'interface détail

Interface Maître-Détail

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



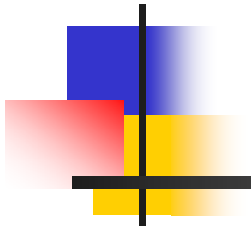
Interface Maître-Détail

Le module maître

- La collection
  - Stocker les éléments de la collection (collection.xml) dans un Flex Module qui sera enregistré dans le dossier components
  - Ce fichier sera utilisé par les composants master et detail
  - Ce 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>
```



Interface Maître-Détail

- Le module maître
  - Pour la création du module maître, nous allons partir sur le composant Box que nous allons modifier afin de pouvoir insérer un DataGrid que nous allons remplir avec les données de la collection
  - On obtient ainsi le fichier master.mxml dans le dossier components/master

```
<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)  vers ici : arrayUser
    <mx:Binding source="col.user" destination="arrayUser.source"/>
     //On lie les données de arrayUser vers 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>
```

Interface Maître-Détail

Le module maître

- Remplissage du DataGrid avec les données de la sélection
  - Afin de remplir le DataGrid, lier les données du modèle avec la propriété dataProvider de DataGrid qui porte l'identifiant masterList :

```
<mx:Binding source="col.user" destination = "detail.dataProvider"/>
```

  - Il va falloir indiquer aux colonnes quels champs du modèle afficher
    - Pour cela, nous allons nous servir de la propriété dataField de la balise <mx:DataGridColumn> :

Interface Maître-Détail

Le module maître

- 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 ajoutera 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
  - Stockées 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
  - Elle fait partie du package components
  - Cette classe 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>
```