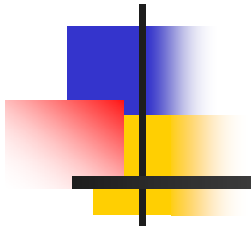
Flex

Création de modèle

ou

Gestion de données sous Flex



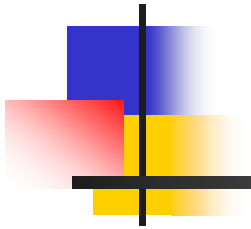
Gérer des données

- Définir un modèle de données
  - Flex propose de définir un modèle de données ou patron pour guider la manipulation et le stockage des données

Gérer des données

- Étape 1
  - Construire un formulaire à partir duquel on entre les données

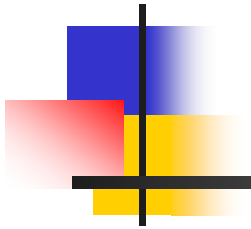
En appuyant sur « Afficher l'utilisateur », les données seront structurées et affichées suivant la structure du modèle



Gérer des données

- Pour construire le formulaire, utiliser :
  - <mx:form> avec :
    - defaultButton :
      - L'attribut pour spécifier le bouton qui validera la formulaire par défaut
    - <mx:FormHeading>
      - Une balise pour afficher un titre
    - <mx:FormItem>
      - Une balise pour décrire chaque champ du formulaire : boîtes de texte, de sélection…

```
<mx:Panel title="Inscription">
<mx:Form defaultButton="{print}">
      <mx:FormHeading label="Détails"/>
      <mx:FormItem>
            <mx:ComboBox id="title">
                  <mx:dataProvider>
                        <mx:String>Mr</mx:String>
                        <mx:String>Mme</mx:String>
                              <mx:String>Melle</mx:String>
                  </mx:dataProvider>
               </mx:ComboBox>
            </mx:FormItem>
      <mx:FormItem label="Prénom">
            <mx:TextInput id="firstname"/>
      </mx:FormItem>
      <mx:FormItem label="Nom">
            <mx:TextInput id="lastname"/>
      </mx:FormItem>
      <mx:FormItem label="NomDate de naissance">
            <mx:TextInput id="birdthday"/>
      </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>
<mx:ControlBar horizontalAlign="center">
<mx:Button id="print" label="Afficher
      l'utilisateur" click="printUser()"/>
</mx:ControlBar>
</mx:Panel>
```



Gérer des données

- Étape 2 : associons maintenant un modèle de données au formulaire
  - Pour créer un modèle de données
    - ❖ On ouvre une balise <mx:Model>
    - ❖ On lui donne un id
    - La balise contient une arborescence XML et contient obligatoirement un nœud racine
    - A l'intérieur, on trouvera les nœuds fils qui décriront le modèle
  - Pour notre utilisateur précédent, on peut définir le modèle suivant

Gérer des données

- Le modèle

```
<mx:Model id="usermodel">
  <user>
        <title/>
        <name>
              <first/>
              <last/>
        </name>
        <bidthday/>
        <phone/>
        <email/>
        <zipcode/>
  </user>
</mx:Model>
```

  - On peut remarquer que ce modèle ne contient aucune donnée
  - Pour l'instant, tous les champs ont la valeur nulle

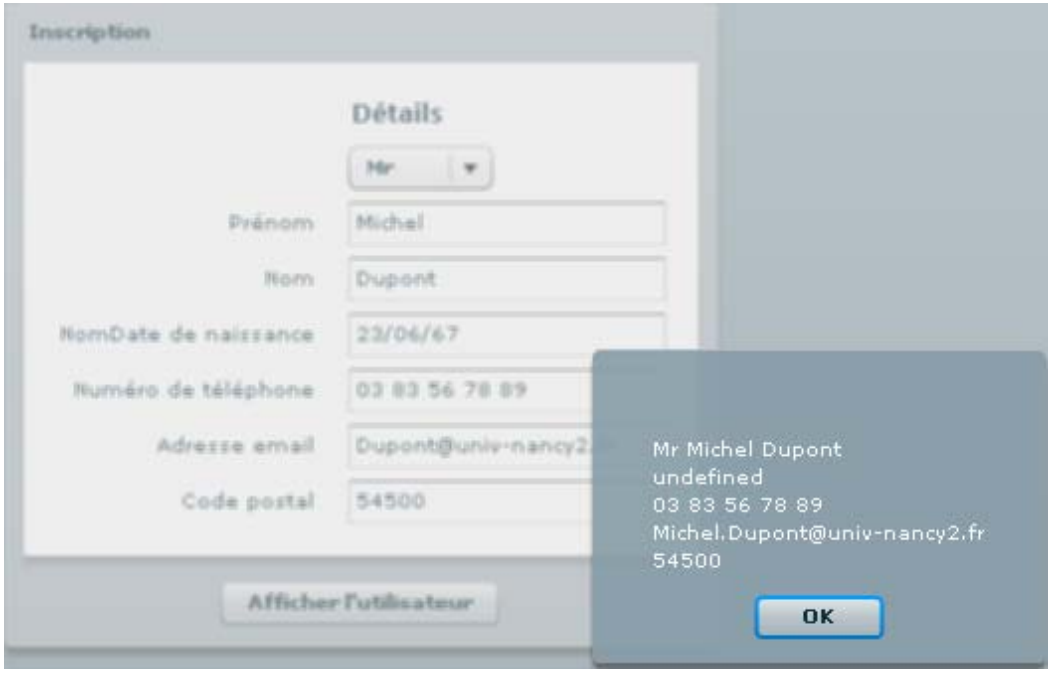
<mx:model>

- Étape 3 : lier le modèle au formulaire
  - ➔ on place entre accolades l'id de l'élément du formulaire qui contient la valeur à utiliser

```
<mx:Model id="usermodel">
   <user>
            <title>{title.value}</title>
            <name>
                     <first>{firstname.text}</first>
                     <last>{lastname.text}</last>
            </name>
            <birdthday>{birdthday.text}</birdthday>
            <phone>{phone.text}</phone>
            <email>{email.text}</email>
            <zipcode>{zipcode.text}</zipcode>
</user>
</mx:Model>
```

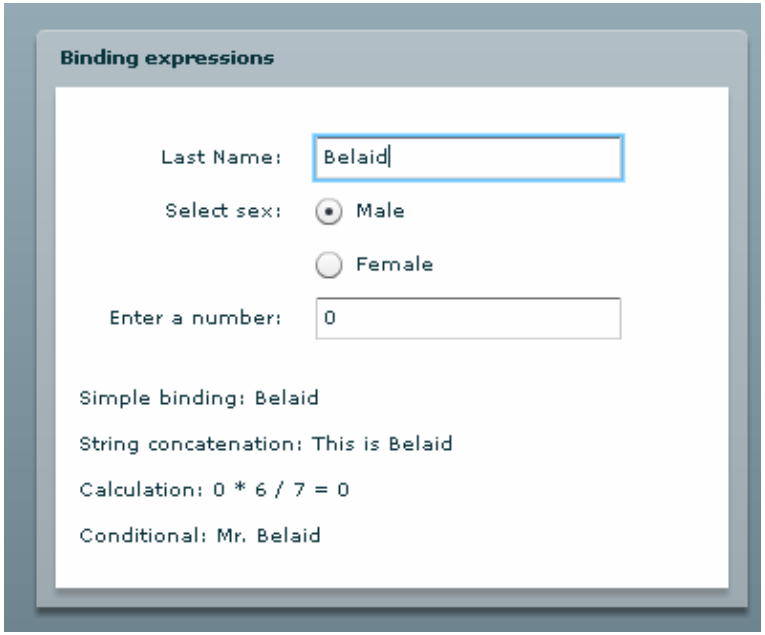
<mx:model>

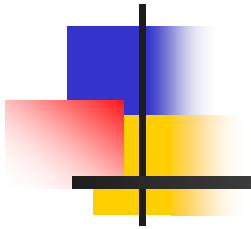
- Résultat : model_binding1.mxml



<mx:model>

- Un autre exemple : model_binding2.mxml
  - Ce qu'on écrit est affiché directement dans les champs correspondants





<mx:XML>

- <mx:XML>
  - À la place de la balise <mx/Model>, on peut utiliser <mx:XML>
  - La seule différence avec la précédente, c'est qu'elle permet d'accéder aux fonctions avancées qui gèrent le XML dans ActionScript3
  - Celles-ci suivent les spécifications d'ECMAScript pour XML
  - Pour s'assurer du respect de ce format, il suffit de définir l'attribut format=e4x

- Exemple :

  Soit le fichier musik.xml

```
<?xml version="1.0" encoding="utf-8"?>
<musik>

    <track trackId="1">
        <genre>Rock</genre>
        <artiste>Mika</artiste>
        <annee>2007</annee>
        <titre>Grace Kelly</titre>
    </track>

    <track trackId="2">
        <genre>Pop</genre>
        <artiste>Lily Allen</artiste>
        <annee>2007</annee>
        <titre>Smile</titre>
    </track>
</musik>
```

- Exemple : XML_binding1.mxml

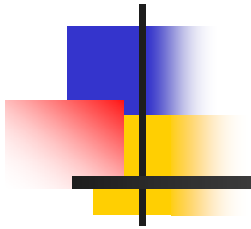
  et le fichier XML_Binding1.mxml

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

   <mx:XML id="xml" source="musik.xml" xmlns="" />

   <mx:ComboBox id="cb1"
         dataProvider="{xml.track.genre}" x="183"
         y="122"/>
   <mx:ComboBox id="cb2"
         dataProvider="{xml.track.artiste}" x="183"
         y="175"/>
   <mx:ComboBox id="cb3" dataProvider="{xml.track.titre}"
         x="183" y="55"/>
   <mx:ComboBox id="cb4"
         dataProvider="{xml.track.annee}" x="183"/>

</mx:Application>
```



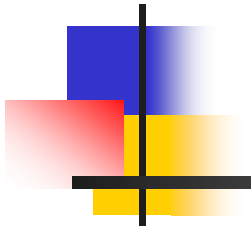
Retour sur <mx:model>

- Comment construire un modèle en passant par des classes ActionScript ?
  - Jusqu'à maintenant, les champs n'étaient que des chaînes de caractères
  - On peut à présent spécifier pour chacun d'eux un type de donnée spécifique
  - Pour cela :
    - On commence par créer un package model (à créer par : File >> new MXML component)
  - Dans ce package,
    - On crée une classe UserModel, dans laquelle on déclare les variables représentant le modèle utilisateur (public)
    - Afin de montrer l'utilité des modèles déclarés dans les classes, on crée une méthode getResume qui va servir à afficher les coordonnées de l'utilisateur

<mx:model>

- Voici la classe UserModel que j'ai rangée dans un répertoire appelé : model
  - UserModel est rangé dans un fichier ActionScript : UserModel.as

```
package model
{
[Bindable]
public var title:String = "Mr";
public var firstname:String;
public var lastname:String;
public var birdthday:String;
public var phone:String;
public var email:String;
public var zipcode:uint;
public function getResume(): String{
    return title + "   "   + firstname + "   "  + lastname +   "\r " +
    birdthday.toDateString() + "\r " + phone + "\r " + email + "\r " +
    zipcode;
}}}
```



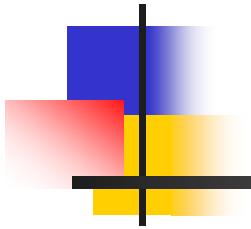
<mx:model>

- Lier le modèle
  - Elle se fait grâce à la méta donnée [Bindable]
  - Elle permet la liaison des données sur toutes les propriétés public de la classe
  - Pour des propriétés private ou protected, il faudrait ajouter la méta donnée à chaque fois, ce qui rendrait l'exemple un peu lourd
- Créer ensuite l'application mxml : Modele1.mxml
  - Déclarez la balise <mx:Application> avec le namespace model
    - <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" xmnls:model="model.*">
  - Toutes les classes du package model seront accessible grâce à ce namespace

<mx:model>

- Faire appel à la fonction getResume() du usermodel :

```
<mx:Script>
<![CDATA[
    import mx.controls.Alert;
    public function printUser():void{
      Alert.show(usermodel.getResume(), "Nouvel
      Utilisateur !");
}
]]>
</mx:Script>
```



<mx:model>

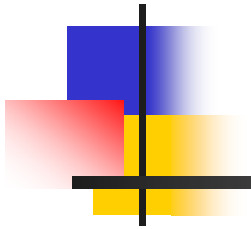
- Accéder au modèle dans l'application
  - Pour cela, on écrit :

```
<model:UserModel id="usermodel"/>
```

- Faire la connexion entre les champs du formulaire et les variables du modèle :
  - Utilise l'attribut change de vos boîtes de dialogue afin de modifier la valeur des propriétés lorsqu'elles ont saisies
  - Exemple :

```
<mx:FormItem label="Nom">
<mx:TextInput id="lastname"
   change="{usermodel.lastname = lastname.text;}"/>
</mx:FormItem>
```

  - Le résultat est dans : Modele1.mxml

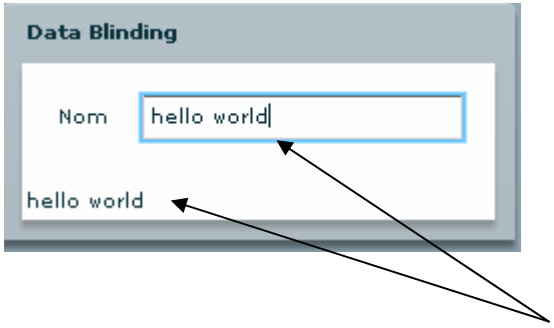


Lier des données

- Objectif
  - Le data binding, ou liaison de données, permet d'automatiser le passage de données entre des parties de votre application
  - Ceci est très utile quand vous avez besoin, par ex. de faire évoluer un élément en fonction de données saisies ou reçues :
    - un graphique qui va se mettre à jour,
    - un texte qui se modifie…

Lier les données

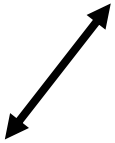
- Liaison par des accolades en MXML
  - C'est le moyen le plus simple d'utiliser la liaison de données (On a déjà vu des exemples)
  - Commençons par un exemple
    - Créer une application contenant un formulaire avec un unique champ de texte qui servira à rentrer un nom
    - Ce champ sera lié avec un label textuel qui se remplira au fur et à mesure de la saisie



Lier les données

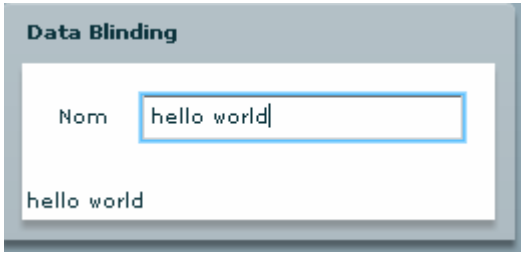
- Exemple : liaison_don1.mxml

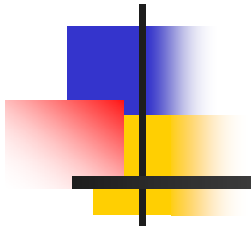
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute">
   <mx:Panel title="Data Blinding">
    <mx:Form>
          <mx:FormItem label="Nom">
                <mx:TextInput id="lastname"/>
          </mx:FormItem>
   </mx:Form>
   <mx:Label id="res" text="{lastname.text}"/>
   </mx:Panel>
</mx:Application>
```



Lier les données

- Les accolades en MXML (suite)
  - Pour utiliser la syntaxe des accolades, il suffit de placer entre elles la source de données et d'affecter le tout en tant que valeur d'une propriété qui sera la destination
  - Notre label res devient donc :
    - <mx:Label id="res" text=" {lastname.text} " />
  - Le texte de ce dernier évoluera automatiquement en fonction des données saisies dans le champ texte
  - Vous pouvez constater le fonctionnement par vous même

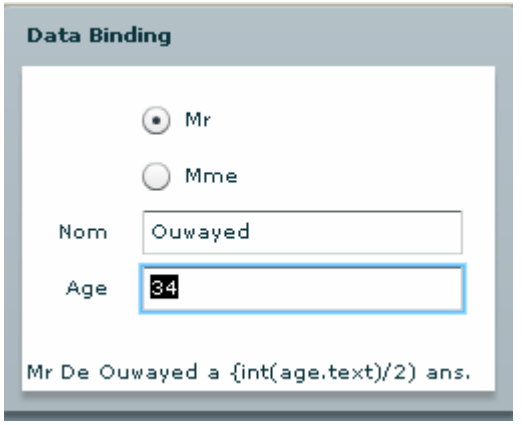


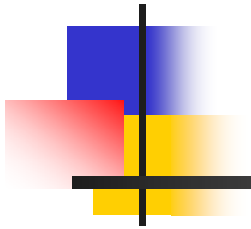


Lier les données

- Utiliser du code ActionScript dans les accolades
  - Il est possible de faire appel à l'ActionScript dans les accolades
  - Il suffit que l'expression renvoie une valeur : expression conditionnelle, concaténation de chaînes, fonctions diverses, …
  - Exemple : liaison_don2.mxml
    - Voici un exemple avec un formulaire et une phrase qui évoluera en fonction de la saisie
    - Cette fois, on fait effectuer la liaison des données dans un data model et on affichera le contenu du modèle
    - Le modèle représente une personne avec son nom, titre…
    - Dans le formulaire, un radiobutton permettra de sélectionner le titre

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
<mx:Model id="personn">
    <personn>
        <title>{man.selected ? "Mr" : "Mme"}</title>
        <name>{"De " + lastname.text}</name>
        <age>{int(age.text)/2)</age>
    </personn>
</mx:Model>
<mx:Panel title="Data Binding">
  <mx:Form>
    <mx:FormItem>
        <mx:RadioButton id="man" label="Mr" groupName="title"
        selected="true"/>
        <mx:RadioButton id="woman" label="Mme"
        groupName="title"/>
    </mx:FormItem>
    <mx:FormItem label="Nom">
        <mx:TextInput id="lastname"/>
    </mx:FormItem>
    <mx:FormItem label="Age">
        <mx:TextInput id="age"/>
    </mx:FormItem>
</mx:Form>
<mx:Label id="res" text="{personn.title + ' ' + personn.name + ' a ' +
    personn.age + ' ans.'}"/>
</mx:Panel>
</mx:Application>
```





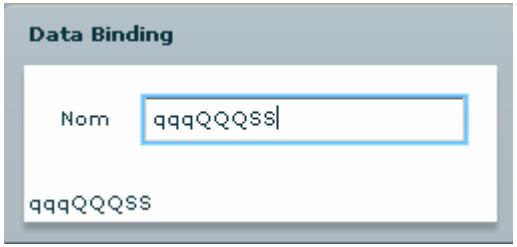
Lier des données

- La balise <mx:Binding> en MXML
  - Au lieu d'utiliser les accolades, on peut faire appel à la balise <mx:Binding>
  - Cette façon de faire est un moyen de respecter l'architecture MVC puisqu'elle sépare totalement la vue du modèle tandis que la balise est le contrôleur
  - Utilisation
    - Lui préciser un attribut source d'où proviennent les données
    - et également un attribut destination qui sera mis à jour

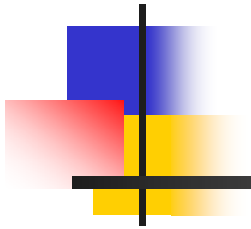
<mx:Binding>

- Exemple : binding0.mxml

```
<mx:Application xmlns:mx=
   "http://www.adobe.com/2006/mxml"
   layout="absolute">
<mx:Binding source="lastname.text"
   destination="res.text"/>
<mx:Panel title="Data Binding">
  <mx:Form>
    <mx:FormItem label="Nom">
            <mx:TextInput id="lastname"/>
    </mx:FormItem>
</mx:Form>
<mx:Label id="res"/>
</mx:Panel>
</mx:Application>
```



  - On constate que le résultat est le même qu'avec les accolades



<mx:Binding>

- Plusieurs variantes
  - On peut avoir ces variantes d'accolades pour le même résultat

```
<mx:Binding source="{'Texte : ' + lastname.text}"
   destination="res.text"/>

<mx:Binding source="'Texte : ' + {lastname.text}"
   destination="res.text"/>
```

<mx:Binding>

- Plusieurs sources ou plusieurs destinations
  - Que ce soit avec les accolades ou avec la balise <mx:Binding>, on peut lier une même source de données à plusieurs destinations
- Une source, deux destinations
  - Pour ajouter une autre destination à l'exemple précédent : ajouter
    - <mx:Binding source="**'Texte : '** + {lastname.text}" destination="res2.text"/>
  - et
    - <mx:Label id="res2"/>
  - Exemple : binding1.mxml
    - On constate que les deux labels évoluent en même temps

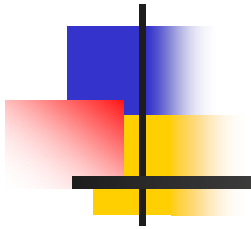
Les deux destinations

<mx:Binding>

- Plusieurs sources, une destination
  - Ce n'est qu'avec la balise <mx:Binding> qu'on peut faire cette association
  - Pour cela
    - Ajouter autant de Binding que de sources
    - Préciser une seule destination
    - C'est la dernière source qui sera prise en compte
  - Exemple : binding2.mxml

```
<mx:Binding source="lastname.text"
   destination="res.text"/>
<mx:Binding source="firstname.text"
   destination="res.text"/>
…
<mx:Label id="res"/>
```

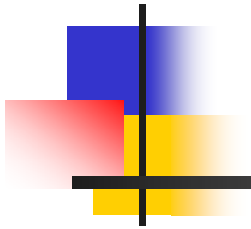
Dernière source



Lier des données

- Lier des données en ActionScript
  - Une dernière méthode permettant de lier les données
    - Il suffit d'utiliser la classe mx.binding.utils.BindingUtils et sa méthode statique bindProperty
    - Il faut lui préciser l'identifiant et l'attribut concernés de la destination et de la source
    - Exemple : binding3.mxml

```
<mx:Script>
<![CDATA[
import mx.binding.utils.BindingUtils;
private function initializeHandler():void
{
    BindingUtils.bindProperty(res, "text", lastname, "text");
}
]]>
</mx:Script>
```



Valider des données

- Principe
  - Les données saisies par l'utilisateur peuvent comporter des erreurs, non correctement formatées…
  - La validation des erreurs est essentielle du côté serveur
  - Mais Flex fournit également des éléments qui permettent une première étape de validation du côté client : les validators
- Les validateurs prédéfinis
  - Flex fournit des validateurs prédéfinis qui permettent de vérifier que les entrées respectent certains critères
    - CreditCardValidator
    - CurrencyValidator
    - DateValidator
    - EmailValidator
    - NumberValidator
    - PhoneNumberValidator…

Valider des données

- Utilisation
  - <mx:Validator> avec le nom du validateur à la place de validator
  - Exemple : validateur1.mxml pour valider le numéro de téléphone

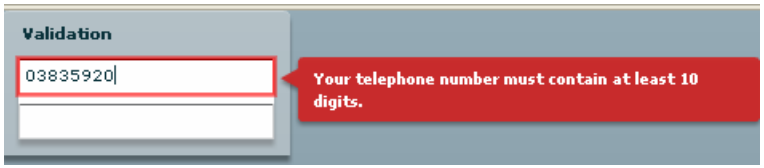
```
<mx:PhoneNumberValidator
   id="phoneValidator"
   source="{phone}" property="text"/>

   <mx:Panel title="Validation">
   <mx:TextInput id="phone"/>
   <mx:TextInput id="nothing"/>
</mx:Panel>
```

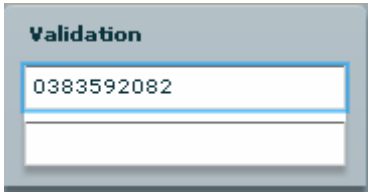
  - Essayer de tromper le validateur
    - Ne rien taper pour commencer
    - Mais sélectionner la seconde boîte de texte
      - La première s'entoure de rouge

Valider des données

- Utilisation
  - Entrez alors un numéro de moins de dix chiffres et changez à nouveau le focus
    - Un nouveau message apparaît



    - A partir du moment où votre numéro fait dix chiffres, il est validé

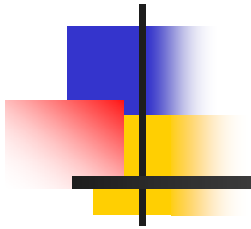


Valider des données

- Validation par ActionScript :
  - Créer le validateur dans une variable
  - Associer les attributs **source** et **property** lorsque la création du composant d'entrée est terminée
  - Exemple : validateur2.mxml

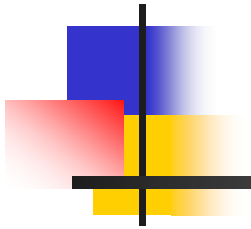
```
<![CDATA[
    import mx.validators.PhoneNumberValidator;
    private var validator:PhoneNumberValidator = new
    PhoneNumberValidator();
private function linkValidator():void
{   validator.source = phone;
     validator.property = "text";}
]]>
</mx:Script>
<mx:Panel title="Validation">
    <mx:TextInput id="phone" creationComplete="linkValidator();"/>
    <mx:TextInput id="nothing"/>
</mx:Panel>
```

  - Le résultat est le même que précédemment



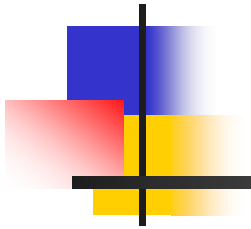
Valider des données

- Utiliser des validateurs dans un formulaire
  - En application à ce qui a été vu, on va utiliser des validateurs prédéfinis pour valider un formulaire d'enregistrement
- Exercice : validateur3.mxml
  - Dans une nouvelle application MXML, créez un formulaire contenant
    - les champs : prénom, anniversaire, numéro de téléphone, adresse e-mail et code postal
  - Ajoutez un bouton à la fin qui sert à valider, activé qu'une fois tous les champs sont valides
  - Créez ensuite les validateurs associés aux champs
  - À l'intérieur des balises de script, créez une variable booléenne, formsValid, initialisée à false qui servira à définir l'état du bouton
  - Permettez qu'elle puisse être liée à une destination grâce à [Bindable]



Valider des données

- Créer ensuite une méthode validateForm qui servira à valider le contenu du formulaire
- Faites appel à cette méthode à chaque fois qu'un changement sera détecté dans un des champs du formulaire à l'aide de l'attribut change des champs de texte
- Cette façon de faire permet de guider plus facilement l'utilisateur en lui précisant au fur et à mesure de la frappe si son entrée est valide



Valider des données

- Voilà à titre d'exemple ce que l'on doit mettre pour le champ birthday :
  - Dans la fonction ValidateForm(event:Event)

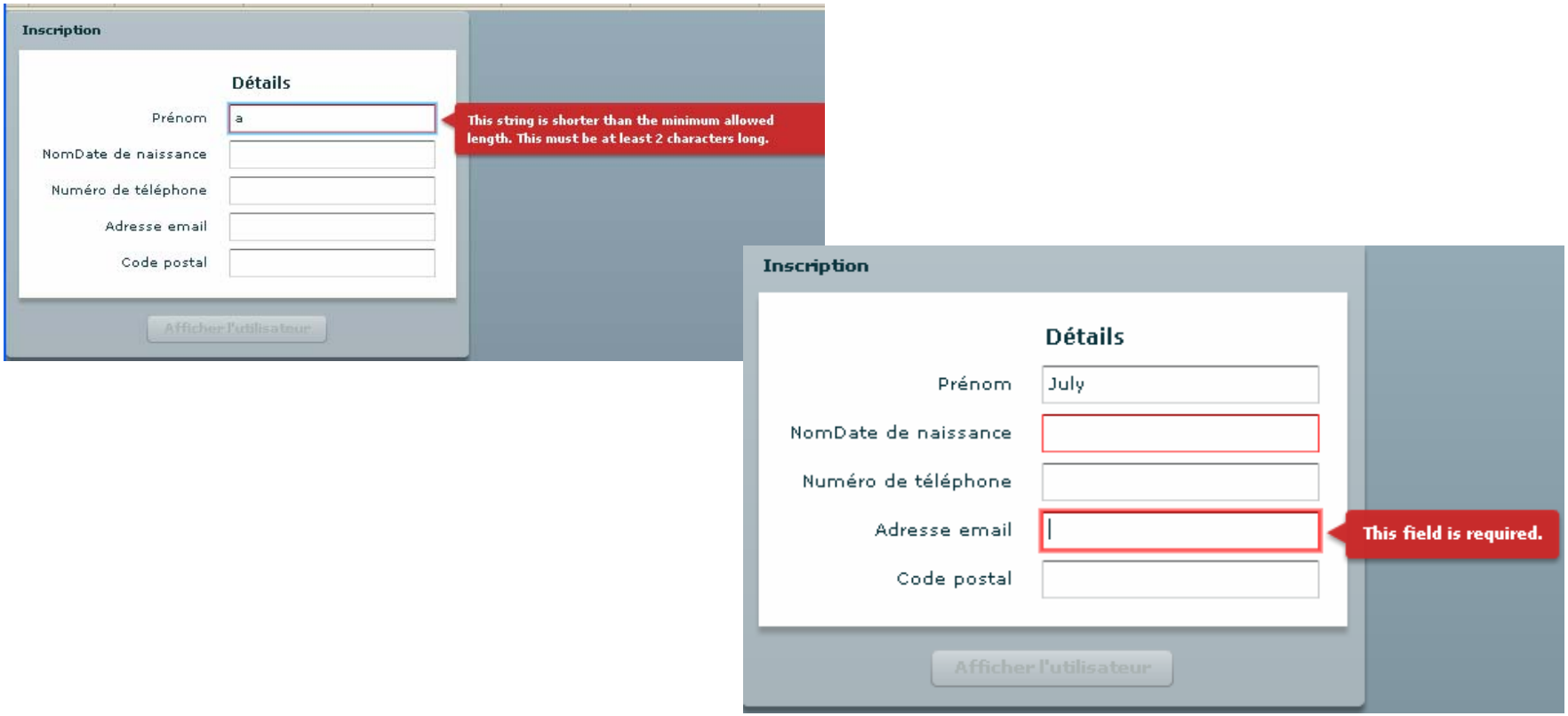
```
if(birthday.text == "")
return;
resultTmp = birthdayvalid.validate();
if(resultTmp.type == ValidationResultEvent.INVALID)
return;
```

  - Et dans le formulaire

```
<mx:FormItem label="NomDate de naissance">
   <mx:TextInput id="birthday" change="validateForm(event);"/>
</mx:FormItem>
```

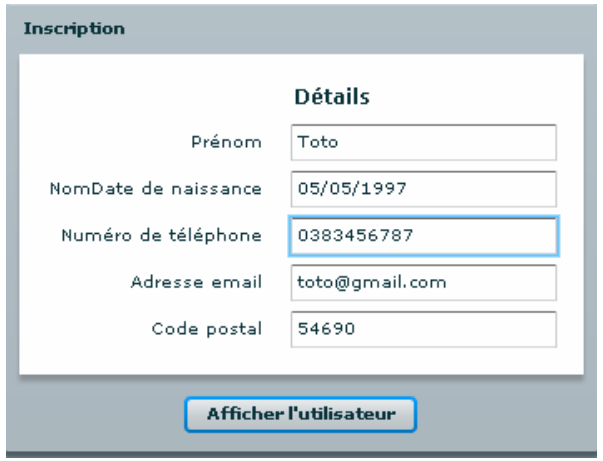
Valider des données

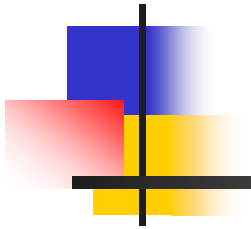
- Voilà ce qu'on devrait avoir pendant le remplissage



Valider des données

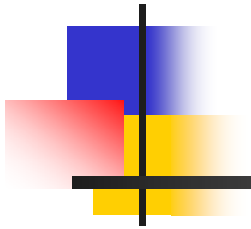
- L'exemple valide les champs dans l'ordre de remplissage
- On utilise à cet effet la méthode validate qui renvoie un ValidationResultEvent qui sert à déterminer le résultat de la validation
- Aucune soumission n'est possible tant qu'un champ n'est pas validé
- Dés que tous sont validés, on peut soumettre le formulaire ➔ le bouton « Afficher l'utilisateur » devient visible





Valider des données

- Personnalisation de la validation
  - On peut créer ses propres validateurs
    - en faisant hériter une classe ActionScript de la classe mx.validators.Validator, ou
    - en étendant les possibilités des validateurs prédéfinis en faisant hériter de la classe appropriée
  - Exemple : validateur4.mxml
    - Créer un nouveau validateur, FullNameValidator, qui va vérifier qu'une chaîne de caractères est bien composée d'au moins deux mots qui seront le nom et le prénom
    - Créer une classe ActionScript de ce nom dans un package components et faites là dériver de la classe Validator en n'oubliant pas de générer le constructeur



Valider des données

- Exemple (suite)
  - À l'intérieur de cette classe, on doit redéfinir la méthode doValidation qui prend en argument l'objet à valider et renvoie un tableau contenant les erreurs de validation

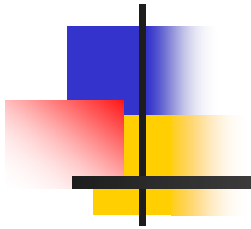
```
package components
{
import mx.validators.Validator;
import mx.validators.ValidationResult;

    public class FullNameValidator extends Validator
    {
        private var results:Array;
        public function FullNameValidator()
        {
                super();
        }
        override protected function doValidation(value:Object):Array
        {
                results =[];
                results = super.doValidation(value);
                if (results.length > 0)
                        return results;
                var baseString:String = String(value);
                var fullName:Array = baseString.split( ' ' );
                if (fullName.length < 2) {
                        results.push(new ValidationResult(true,
                        null, "notfullName", "Vous devez entrer un nom et
                        prénom."));
                        return results;
                }
                return results
        }
    }
}
```

Valider les données

- Validation personnelle (suite)
  - ValidationResult permet de définir une erreur lors de la validation en spécifiant le message d'erreur
  - Voici l'appli totale


```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
   layout="absolute" xmlns:Comp="components.*">
   <Comp:FullNameValidator source="{fullName}" property="text"/>
   <mx:Panel title="Validation">
            <mx:TextInput id="fullName"/>
            <mx:TextInput id="nothing"/>
   </mx:Panel>
</mx:Application>
```



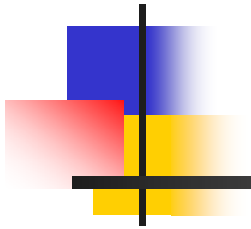
Formater les données

- Principe
  - Flex fournit des formateurs pour un affichage standard des données saisies
  - Ces formateurs déclenchent la mise en forme des données avant qu'elles ne soient affichées
- Utiliser les formateurs de base
    - CurrencyFormatter
    - DateFormatter
    - NumberFormatter
    - PhoneFormatter
    - ZipCodeFormatter
  - Toutes ces classes dérivent de mx.formatters.Formatter

Formater les données

- Exemple : formateur1.mxml

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute">
<mx:CurrencyFormatter id="formatter"
    precision="2"
    alignSymbol="right"
    decimalSeparatorTo="."
    thousandsSeparatorTo=" "
    currencySymbol=" "
    rounding="nearest"/>
<mx:Panel title="Formatage de données">
<mx:TextInput id="currency"/>
<mx:Label id="FormattedCUrrency" text="Prix :
    {formatter.format(currency.text)}"/>
</mx:Panel>
</mx:Application>
```



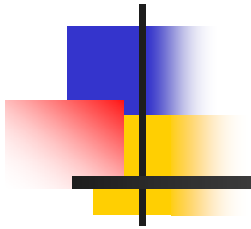
Formater les données

- Récupérer les erreurs de formatage
  - En cas d'erreur lors de la saisie, lorsque la chaîne est impossible à formater, une erreur se produit
  - La raison de cette erreur est obtenue dans l'attribut error
  - Pour observer ce message, associez une fonction printError qui affichera une alerte

Formater les données

- Récupérer les erreurs de formatage
  - Exemple : formateur2.mxml

```
<mx:Script>
<![CDATA[
import mx.controls.Alert;
public function printError():void{
   Alert.show(formatter.error, "Erreur du
   formateur");
}
]]>
</mx:Script>
<mx:CurrencyFormatter …
<mx:Button id="prError" label="Affiche
   l'erreur" click="printError();"/>
```

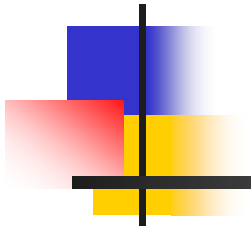


Travailler avec XML

- Avec ActionScript
  - ActionScript 3 fournit des outils basés sur ECMAScript for XML (E4X) qui permettent le traitement simple et puissant de XML
  - Accéder aux données
    - On peut déclarer des variables contenant des données XML en ActionScript
      - En effet XML est un type de données natif

- Exemple : créons la liste suivante : xmldonne1.mxml

```
[Bindable]
private var userList:XML=
<users>
   <user id="1">
            <name>
                     <firstname>Edouard</firstname>
                     <lastname>Ruiz</lastname>
            </name>
            <age>23</age>
            <email>toto@gmail.com</email>
   </user>
   <user id="2">
            <name>
                     <firstname>Vianney</firstname>
                     <lastname>Baron</lastname>
            </name>
            <age>21</age>
            <email>titi@gmail.com</email>
   </user>
</users>
```



Travailler avec XML

- Accéder aux éléments
  - Accéder par l'indice dans une liste : le point « . »
    - userList.user
      - Donne accès à la liste des nœuds user
    - userList.user[i].age
      - Permet d'accéder à l'âge du nœud d'indice i
  - Accéder par l'attribut : « @ »
    - userList.user.(@id==2).name.firstname
      - Permet d'avoir le prénom de l'utilisateur d'identifiant 2

Travailler avec XML

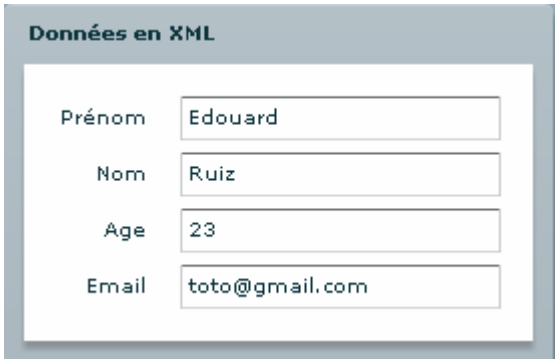
- Exemple

```
<mx:Panel title="Données en XML">
<mx:Text id="all" text="{'Objet XML :\r'+userList.user}"/>
<mx:Label id="age" text="{'Age de l\'utilisateur en position 0 :' + userList.user[0].age}"/>
<mx:Label id="firstname" text="{'Prénom de l\'utilisateur ayant l\'id 2 :' + userList.user.(@id==2).name.firstname}"/>
</mx:Panel>
```

Travailler avec XML

- Modifier les données : xmldonnee2.mxml
  - Conservez la variable userList définie précédemment mais remplacez les labels par ce formulaire

```
<mx:Form>
    <mx:FormItem label="Prénom">
            <mx:TextInput
            id="firstname"
            text="{userList.user[0].name.firstname}"
            change="{userList.user[0].name.
            firstname = firstname.text}"/>
    </mx:FormItem>
    <mx:FormItem label="Nom">
            <mx:TextInput
            id="lastname"
            text="{userList.user[0].name.lastname}"
            change="{userList.user[0].name.lastname = lastname.text}"/>
    </mx:FormItem>
    <mx:FormItem label="Age">…
```



Travailler avec XML

- Exercice : xmldonnee3.mxml
  - Utiliser une DataGrid pour naviguer dans cette liste et ainsi pouvoir modifier tout son contenu

Se reporte là

Ce qu'on modifie ici