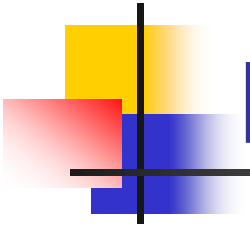
Flex 4.5 / Flash Builder 4.5

Prise en main



Flex / Flash Builder

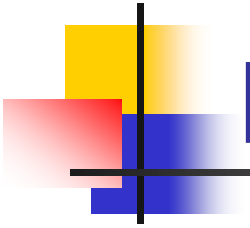
- C’est quoi ces deux noms ?
  - Flex est le framework qui permet de créer des applications web basées sur la technologie Flash
  - Sa prise en main est facilitée par l’outil de développement Flash Builder basé sur Eclipse

Flex / Flash Builder

- L'évolution de Flex : rapide, une version par an, depuis 2004
  - 2004 : Macromedia Flex Server 1.0 et 1.5
    - Ciblé le développement d'application d'entreprise
    - La compilation des SWF se faisait au niveau serveur
  - 2005 : Adobe Flex 2
    - SDK Flex 2 avec un compilateur (mxmlc) et une bibliothèque de composants graphiques et utilitaires
    - Adobe a basé sa nouvelle version de Flex Builder sur la plate-forme Eclipse

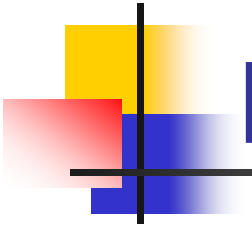
Flex / Flash Builder

- 2007 : Adobe Flex 3
  - Sortie du SDK Flex 3 sous licence Mozilla Public License ainsi qu'Adobe AIR 1.0, sa solution RDA
  - Adobe a sorti le nouveau Flex Builder 3, toujours basé sur Eclipse, permettant de développer des applications Flex et Air
- 2010 : Adobe Flex 4
  - Nouvelles bibliothèque :
    - Spark : composants personnalisables
    - FXG : dessins graphiques
- 2011 : Adobe Flex 4.5 :
  - Flex Hero (Version bêta) : applications mobiles pour Google Android, Apple iOS, BlackBerry Tablet OS



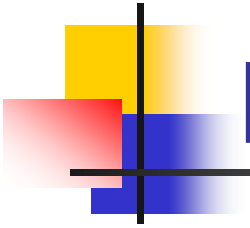
Flex 4.5 / Flash Builder 4.5

- Objectif du cours
  - Comprendre Flex 4.5 par la pratique
  - et aussi, si on a le temps
    - BlazeDS : pour la communuication avec les méthodes Java et le *push de données du LiveCyle*
    - Flash Catalyst : import d'objets développés dans Photoshop…
  - Prendre en main l'outil FlashBuilder 4.5
  - S'amuser :-)
  - Nous utiliserons FlashBuilder 4.5 (Mai 2011)



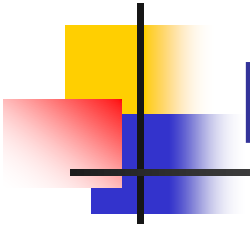
Programme

- Créer des applications Flex 4.5
  - MXML AS3, Binding, composants personnels positionnement, déploiement
- Alimenter Flex en données
  - Passerelle avec PHP : protocole AMF (Action Message Format) et la bibliothèque Zend_AMF
  - Passerelle avec Java : LiveCycle
- Créer des applications pour les mobiles
  - Ergonomie, navigation, déploiement



Flash Builder 4.5

- Installer Flash Builder 4.5
  - Flash Builder est disponible sous Mac et sous Windows
- Flash Builder 4.5 et Linux
  - Contrairement au SDK, la version commerciale de Flash Builder 4 n'est malheureusement pas compatible avec Linux
  - Néanmoins, Adobe a développé le projet Flash Builder Public Alpha, qui consiste en un plug-in à ajouter à l'IDE Eclipse et offrant notamment les fonctionnalités de coloration syntaxique, de compilation, de débogage…
  - Ce plug-in est téléchargeable à l'adresse suivante : *http://labs.adobe.com/technologies/flex/ flexbuilder_linux/*



Flash Builder 4.5

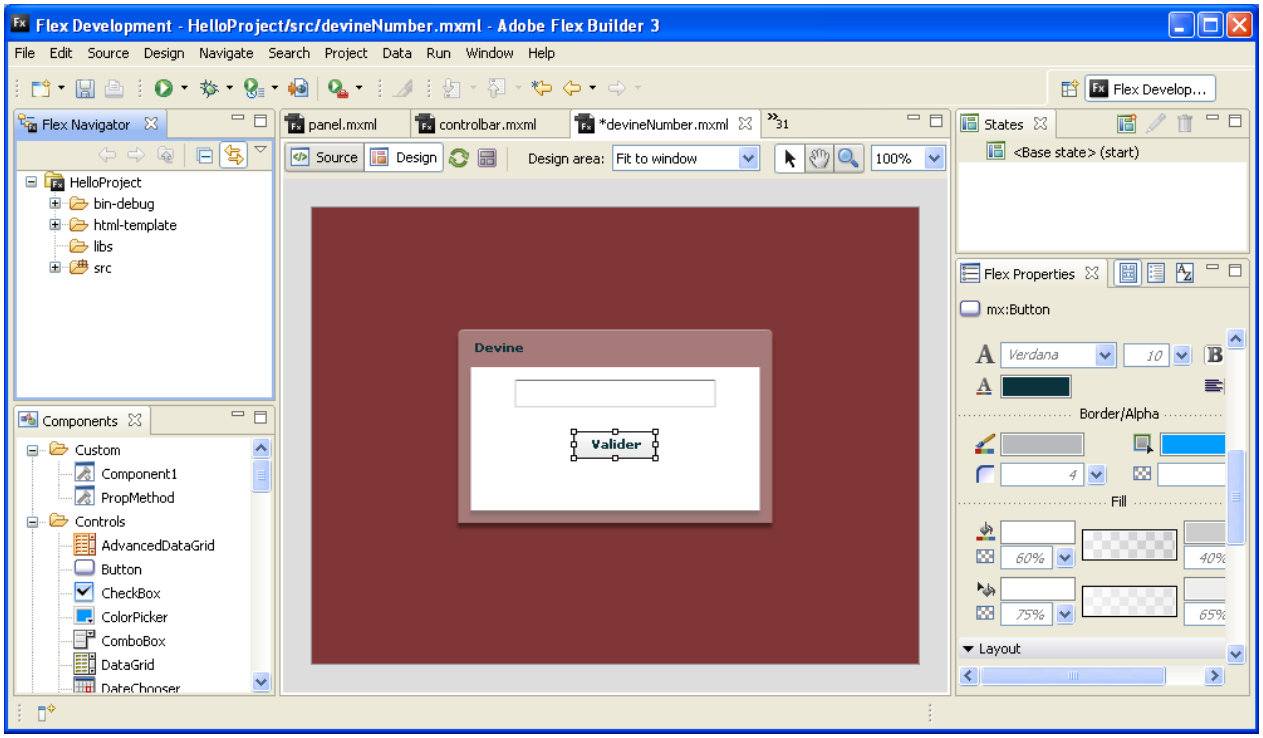
- Installer Flash Builder 4.5 en mode plug-in
  - Installer Eclipse si ce n'est pas fait
  - Rendez-vous ici : http://www.adobe.com/fr/products/flash-builder.html
  - Cliquez sur le bouton [Version d'Evaluation]
  - Une fois l'archive récupérée et décompressée aller dans le dossier *Adobe Flash Builder 4 Standalone*, et lancer l'application *Install* et laissez-vous guider par l'assistant d'installation
  - Adobe en profite pour vous installer Flash Player 10, puisque c'est la version minimum pour exécuter du code Flex 4
  - Une fois l'installation terminée, lancez l'application et poursuivez la configuration

Prise en main

- Au chargement, on doit voir apparaître l'écran suivant en mode création (design):

Navigateur (Arborescence du projet)

Liste des Composants Flex

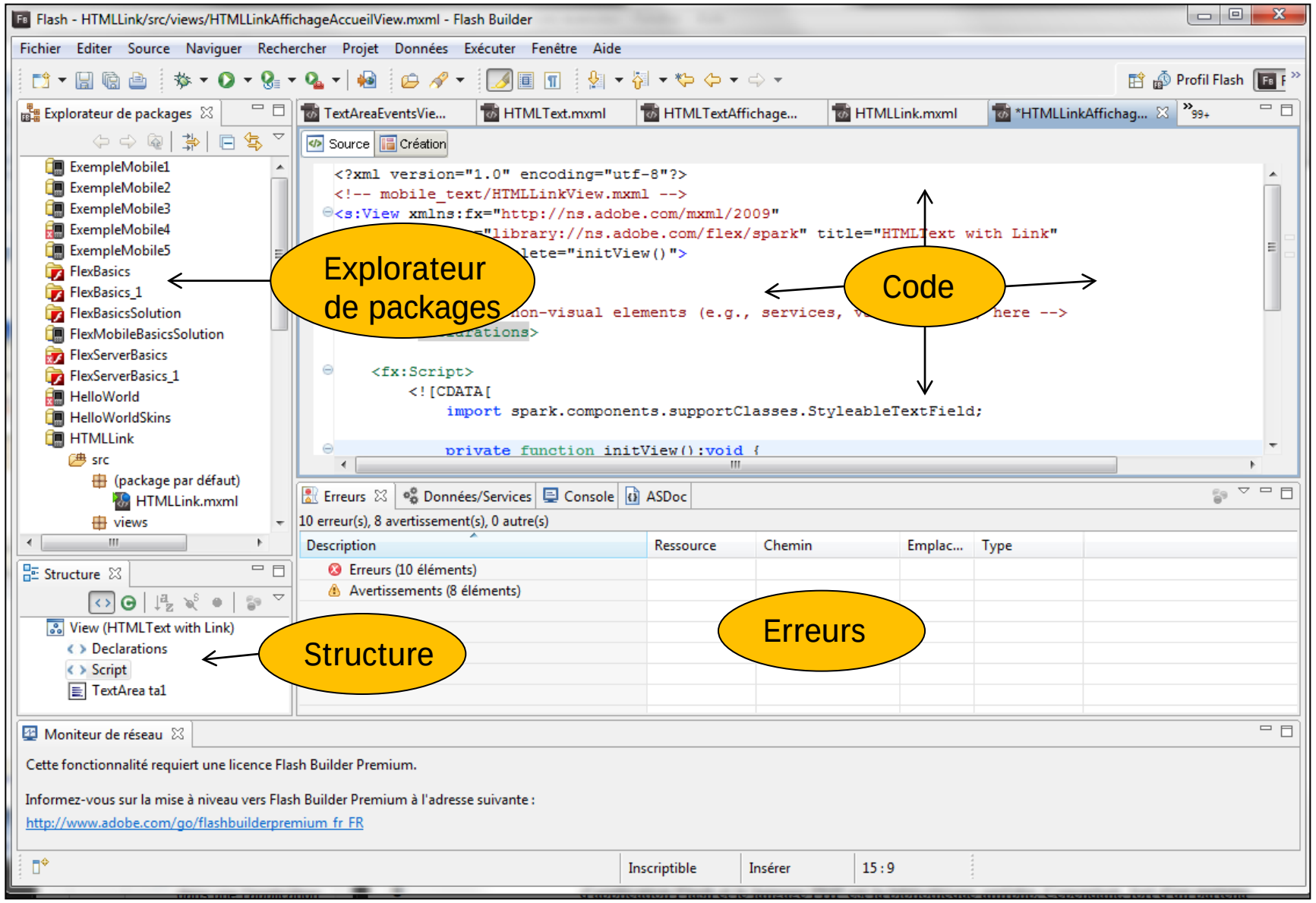


Vue des états

Propriétés du composant sélectionné

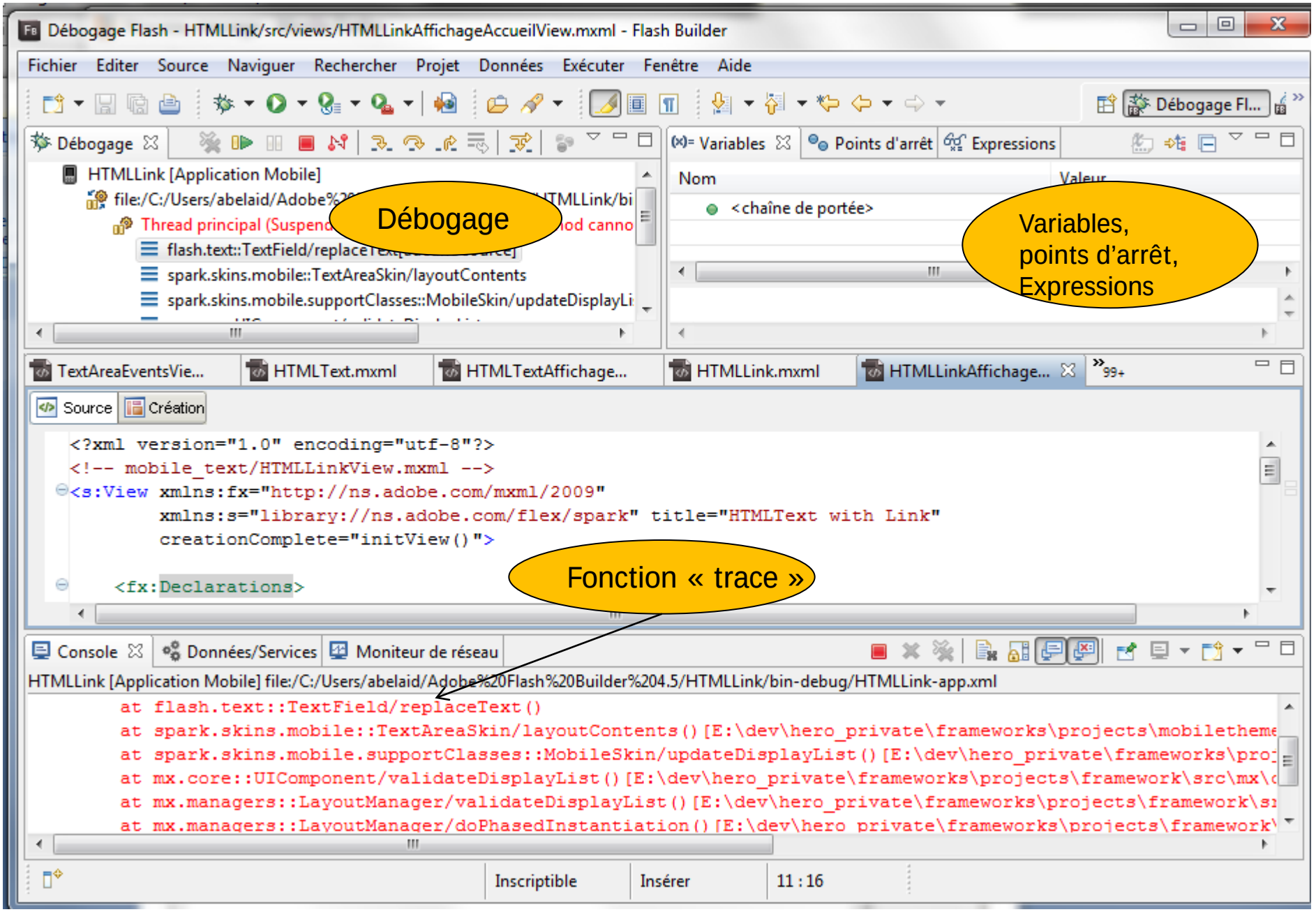
9

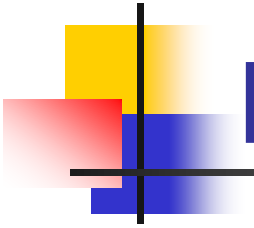
Le mode source



Débogage

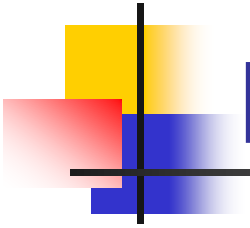
- Fonctionnement
  - Menu : Exécuter >> Débogage
  - Erreur
    - Il arrive que Flash Builder déclenche un message d'erreur lors de l'exécution de la vue de débogage
    - Cela est sans doute dû à la non présence du débogueur du Flash Player 10
    - Dans ce cas, télécharger la version du débogueur associé à votre navigateur disponible en téléchargement à cette adresse : *http://www.adobe.com/support/flashplayer/downloads.html#fp10*


Débogage Flash - HTMLLink/src/views/HTMLLinkAffichageAccueilView.mxml - Flash Builder
Fichier Editer Source Naviguer Rechercher Projet Données Exécuter Fenêtre Aide
Débogage
HTMLLink [Application Mobile]
flash.text::TextField/replaceText
spark.skins.mobile::TextAreaSkin/layoutContents
spark.skins.mobile.supportClasses::MobileSkin/updateDisplayLi
Variables
Points d'arrêt
Expressions
Nom
Valeur
<chaîne de portée>
Débogage
Variables, points d'arrêt, Expressions
TextAreaEventsVie...
HTMLText.mxml
HTMLTextAffichage...
HTMLLink.mxml
HTMLLinkAffichage...
Source
Création
<?xml version="1.0" encoding="utf-8"?>
<!-- mobile_text/HTMLLinkView.mxml -->
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark" title="HTMLText with Link"
creationComplete="initView()">
<fx:Declarations>
Fonction « trace »
Console
Données/Services
Moniteur de réseau
HTMLLink [Application Mobile] file:/C:/Users/abelaid/Adobe%20Flash%20Builder%204.5/HTMLLink/bin-debug/HTMLLink-app.xml
at flash.text::TextField/replaceText()
at spark.skins.mobile::TextAreaSkin/layoutContents()[E:\dev\hero_private\frameworks\projects\mobiletheme
at spark.skins.mobile.supportClasses::MobileSkin/updateDisplayList()[E:\dev\hero_private\frameworks\proj
at mx.core::UIComponent/validateDisplayList()[E:\dev\hero_private\frameworks\projects\framework\src\mx\c
at mx.managers::LayoutManager/validateDisplayList()[E:\dev\hero_private\frameworks\projects\framework\s
at mx.managers::LayoutManager/doPhasedInstantiation()[E:\dev\hero_private\frameworks\projects\framework\
Inscriptible
Insérer
11 : 16



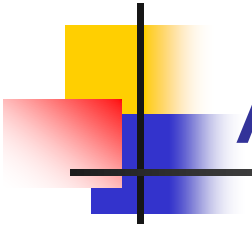
Débogage

- Fonctionnement
  - La vue de débogage affiche les différentes tâches (thread/processus) réalisées par l'application
  - Cette vue permet de suspendre ou de reprendre l'exécution d'une tâche précédemment suspendue afin d'analyser l'exécution de l'application
- Multithreading
  - Une application Flex n'est pas multithread (multitâches)
  - ➔ Par conséquent, l'implémentation d'une barre de progression permettant d'afficher l'état d'avancement de l'exécution d'un traitement n'est pas envisageable



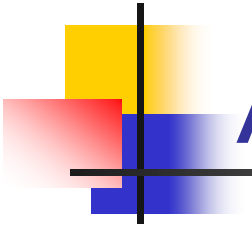
Débogage

- Pour stopper l'exécution à un point précis
  - placer des points d'arrêt dans le code applicatif
- Pour « tracer » son passage
  - utiliser la fonction trace du langage ActionScript
- La vue Console
  - permet de visualiser l'ensemble du traçage réalisé
- La vue Variables
  - affiche les valeurs prises par les différentes variables déclarées
- La vue Expressions
  - permet de surveiller les variables jugées critiques pour l'application



Affichage

- Affichage des différentes vues
  - Il se peut que l'ensemble des vues ne soit pas affiché par défaut
    - Dans ce cas, il suffit de sélectionner le menu Fenêtre et de choisir la vue désirée parmi la liste proposée
  - Néanmoins
    - lors de l'exécution du projet en mode Débogage, Flash Builder vous demandera si vous souhaitez ou non utiliser la perspective de débogage

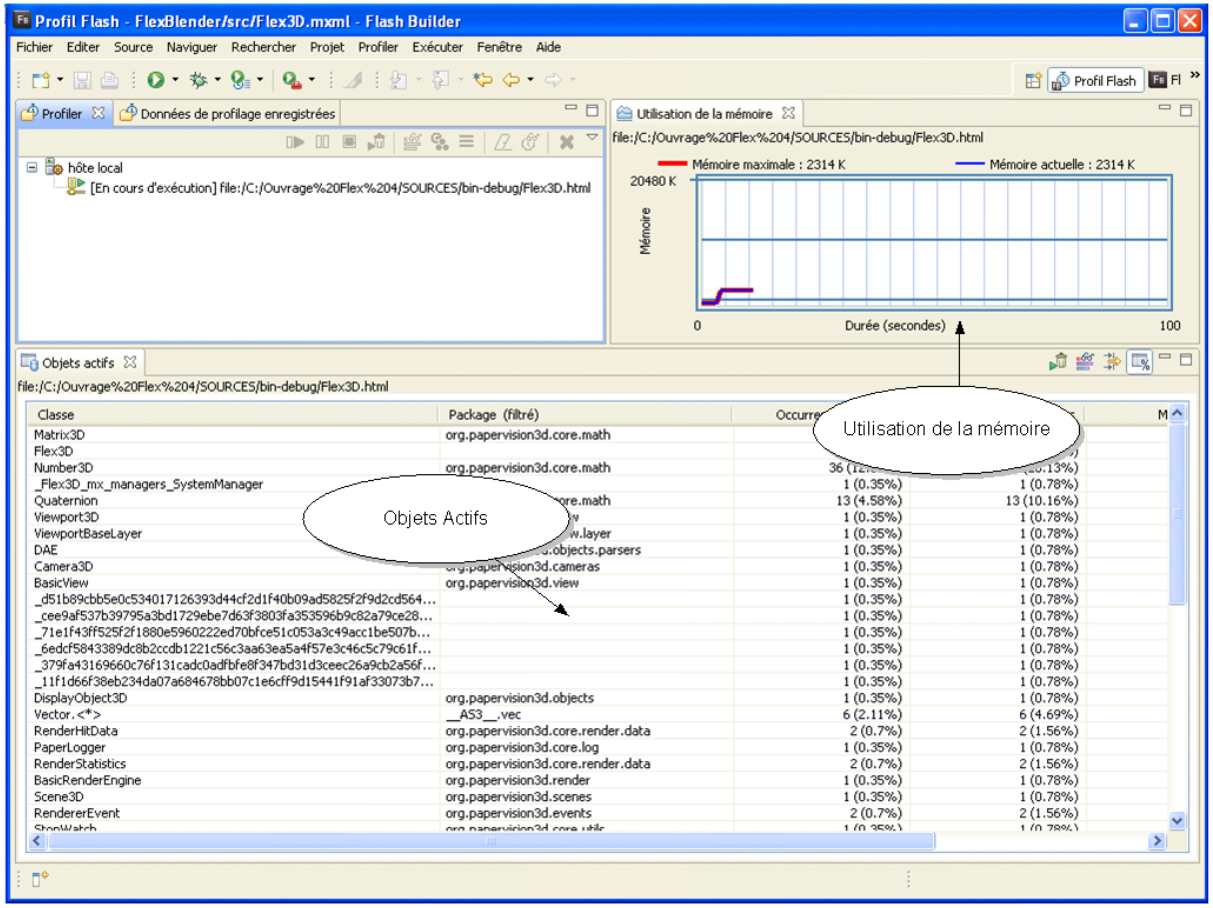


Affichage

- La perspective de profiling
  - Disponible uniquement pour la version Premium
  - Permet
    - d'effectuer une analyse statistique du nombre d'objets instanciés, de la place mémoire qu'ils occupent, des méthodes exécutées et de leur temps d'exécution
  - Comprend deux vues essentielles :
    - la vue Objets Actifs, qui permet de visualiser les objets créés et les méthodes exécutées
    - la vue Utilisation de la mémoire, dédiée à la visualisation du taux d'utilisation de la mémoire

Affichage

- **La perspective de profiling**



Programmation sous Flex

- Explorer mon premier projet
  - Commencez par créer un nouveau projet flex

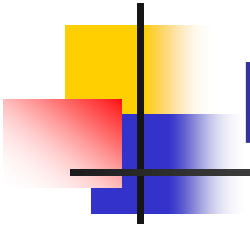
    ➔ File >> New >> Flex project
    - Nommez le par exemple "Chapitre1"
  - Placez-vous sur le répertoire src et créez une application MXML

    ➔ File >> New >> MXML Application
    - Nommez là : "HelloWorld"
    - Une fenêtre s'ouvre avec un code minimal
    - L'enregistrement donne le fichier : HelloWorld.mxml

Programmation sous Flex

- En mode Source : premières lignes de votre page .mxml :

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application
    xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx" minWidth="955"
    minHeight="600">
    <fx:Declarations>
      <!-- Placer ici les éléments non visuels (services et objets de valeur,
      par exemple). -->
    </fx:Declarations>
</s:Application>
```



Programmation sous Flex

- Les espaces de noms
  - Bien qu'il conserve les classes Flex 3 au sein des mêmes packages mx.*, le kit SDK de Flex 4 inclut de nouveaux packages nommés spark.* pour les composants, les classes principales, les effets, les filtres, les mises en forme, les primitives, les habillages et les utilitaires.
  - Le kit SDK de Flex 4 offre un nouvel ensemble de composants et d'effets dont les noms de classe sont souvent identiques à ceux des composants Flex 3
  - Pour éviter les conflits dans MXML, le kit SDK de Flex 4 fournit quatre espaces de nom distincts :
    - MXML 2006
    - MXML 2009
    - Spark
    - Fx

Programmation sous Flex

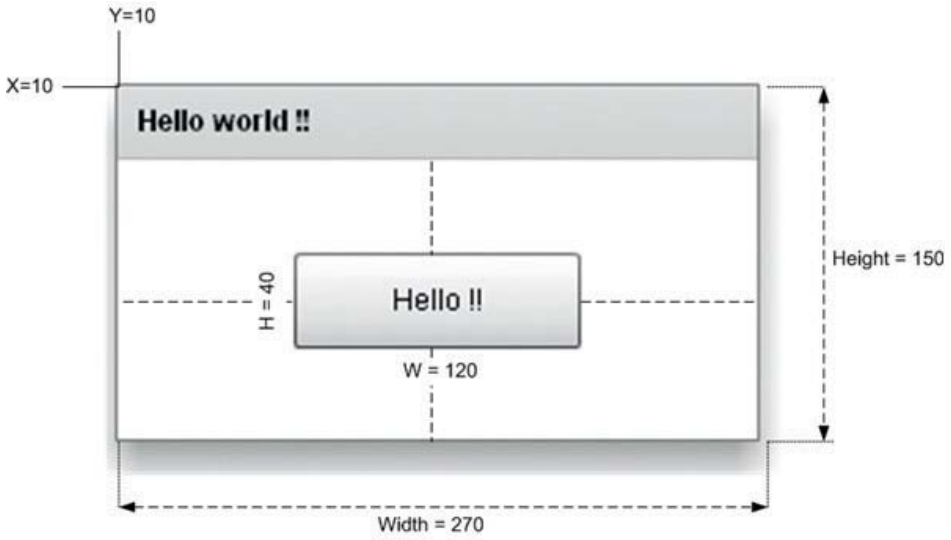
- On peut ainsi mélanger les espaces :

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx">

    <fx:Script>
        <![CDATA[

        ]]>
    </fx:Script>

    <mx:Button label="Halo"/>
    <s:Button label="Spark"/>

</s:Application>
```

Note : il existe 3 namespace : “s” pour les composants Spark, “mx” pour les composants MX et “fx” pour les éléments du langage (Script, Style...)

Programmation sous Flex

- Première application
  - Créez une scène avec un bouton au centre, de la taille indiquée, contenant la chaîne de caractères : « Hello !! »



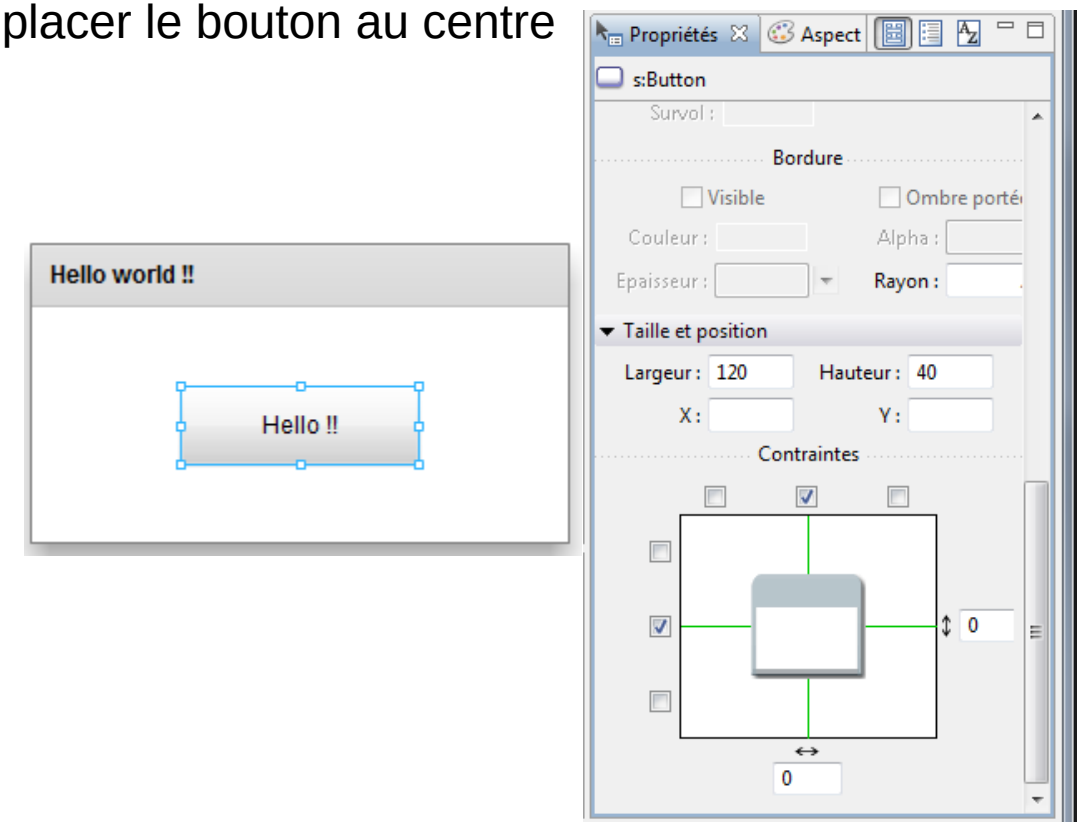
Programmation sous Flex

- Première application
  - Le code correspondant est :

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  xmlns:mx="library://ns.adobe.com/flex/halo" minWidth="1024"
   minHeight="768">
<!-- Ajout du panneau à l'aide de la bibliothèque Spark-->
<s:Panel x="10" y="10" width="270" height="150" id="panneau"
   title="Hello world !!">
<!-- Ajout du bouton -->
<s:Button label="Hello !!" id="bouton_action" height="40" width="120"
   horizontalCenter="0" verticalCenter="0"/>
</s:Panel>
</s:Application>
```

Programmation sous Flex

- En mode Design
  - Cliquez sur le mode « création » ou « design »
  - Observez les propriétés et surtout les contraintes pour placer le bouton au centre



Programmation sous Flex

- Écriture du fichier ActionScript
  - Tout d'abord, créez un nouveau fichier nommé HelloWorldAS.as dans le répertoire src de l'application qui va contenir la procédure permettant d'afficher le message
    - Fichier >> Nouveau >> Fichier ActionScript
  - Saisissez ensuite le code suivant dans ce fichier :

```
import mx.controls.Alert;
    public function afficherMessage():void
            {
                    Alert.show("Hello World !!!");
            }
```

  - On importe la bibliothèque nécessaire à l'utilisation de la classe Alert
  - afficherMessage() fait appel à la méthode show de la classe Alert

Programmation sous Flex

- Liaison graphique et action
  - Nous possédons à présent
    - un fichier MXML servant à la description de l'interface graphique
    - Et un fichier ActionScript permettant d'afficher le message « Hello World !!! »
  - L'étape suivante va consister à créer la liaison entre ces deux fichiers et à définir l'action qui fera appel à la fonction afficherMessage()
  - Cette liaison s'effectue dans le fichier MXML à l'aide de la

balise <fx:Script> propre à l'espace de noms MXML 2009, pour laquelle il suffit de spécifier la source du script à utiliser, c'est-à-dire le chemin relatif du fichier ActionScript :

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

Programmation sous Flex

- Liaison graphique et action (suite) : HelloWorld2.mxml
  - On souhaite que le message s'affiche lorsque l'utilisateur clique sur l'unique bouton de l'interface
  - Cela se fera via la définition de l'action sur click du bouton faisant appel à la fonction afficherMessage()

```
<s:Button label="Hello !!" id="bouton_action"
  height="40" width="120" y="37" x="75"
  horizontalCenter="0" verticalCenter="0"
  click="afficherMessage()"/>
```

Programmation sous Flex

- Compiler une application Flex
  - En appuyant sur la flèche verte, Flex Builder 3 se charge de compiler votre application et de la lancer dans le navigateur par défaut
  - Les étapes
    - Étape 1 :
      - Transforme les balises MXML en classe ActionScript 3
    - Étape 2 :
      - Le compilateur crée du code instanciant cette classe qui sera finalement compilée en fichier SWF (format standard du Flash Player)
  - Le SDK permet de faire ces opérations de manière transparente, mais sinon, on peut utiliser des commandes en ligne et lancer le compilateur de MXML : mxmlc
  - 3 fichiers sont générés dans le répertoire/bin de votre projet :
    - Une version standard pour la mise en production
    - Une version Debug
    - Une version Profile

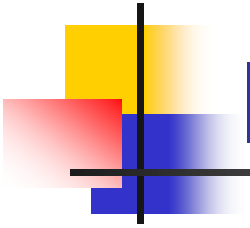
Programmation sous Flex

- Intégration dans une page web : index.html
  - Pour intégrer notre application dans une page web, que nous nommerons index.html
    - utiliser le langage HTML et ses balises <object> qui nous permettront de faire référence au fichier SWF

```
<html>
<head>
<title>Hello World !</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
<p>
<object type="application/x-shockwave-flash" data="HelloWorld.swf" width="100%"
    height="100%">
<param name="movie" value="HelloWorld.swf">
</object>
</p>
</body>
</html>
```

Flash Builder

Réaliser le design des interfaces



Introduction

- Objectif
  - Apprendre à personnaliser le design des interfaces graphiques
    - Découvrir comment positionner et dimensionner les composants de Flex
    - Aborder la notion de style
    - Réaliser des habillages personnels de composants

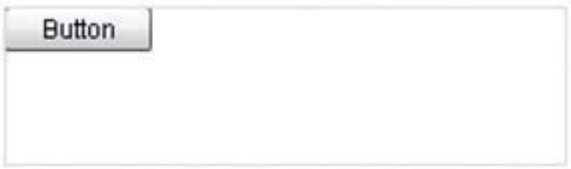
Dimensionner et positionner des composants

- Objectif
  - Une des premières étapes de conception d'interface
- Spécifier les dimensions des composants
  - Plusieurs méthodes
    - Laisser le framework Flex proposer une taille par défaut pour chaque composant inséré dans l'interface
    - Préciser manuellement la taille en pixels de chaque composant
    - Spécifier les dimensions des composants en pourcentage des dimensions de leur conteneur

Dimensionner et positionner des composants
Spécifier les dimensions des composants

- Dimensionnement par défaut
  - Exemple

```
<!-- Conteneur VGroup -->
<s:VGroup x=”59” y=”23” width=”267” height=”75”>
   <!-- Composant : Button -->
   <s:Button id=”btn” label=”Button”/>
</s:VGroup>
```



Dimensionner et positionner des composants
Spécifier les dimensions des composants

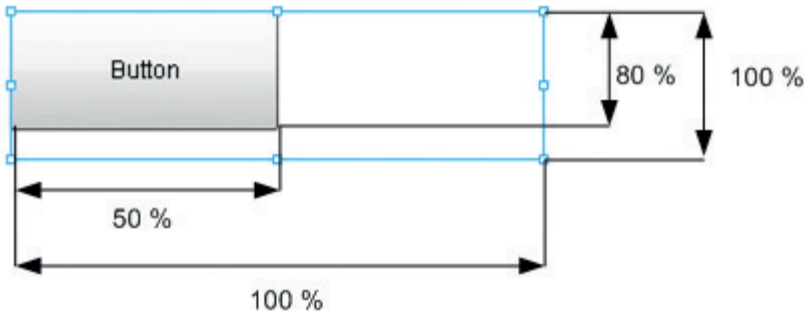
- Spécification des dimensions
  - Exemple

```
<!-- Conteneur VGroup -->
<s:VGroup x=”59” y=”23” width=”267” height=”75”>
<!-- Composant : Button -->
<s:Button id=”btn” label=”Button” width=”114”
   height=”34”/>
</s:VGroup>
```

Dimensionner et positionner des composants
Spécifier les dimensions des composants

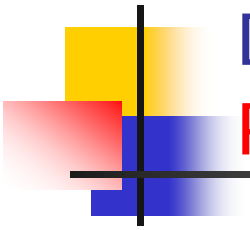
- Utilisation des pourcentages
  - Exemple : on spécifie les dimensions du bouton en fonction de la taille de son conteneur

```
<!-- Conteneur VGroup -->
<s:VGroup x=”59” y=”23” width=”267” height=”75”>
<!-- Composant : Button -->
<s:Button id=”btn” label=”Button” width=”50%”
   height=”80%”/>
</s:VGroup>
```



Dimensionner et positionner des composants

- Positionner les composants
  - Trois méthodes pour placer un composant à un endroit précis de l'interface
    - de manière automatique
    - de manière absolue
    - à l'aide de contraintes (ancrages)



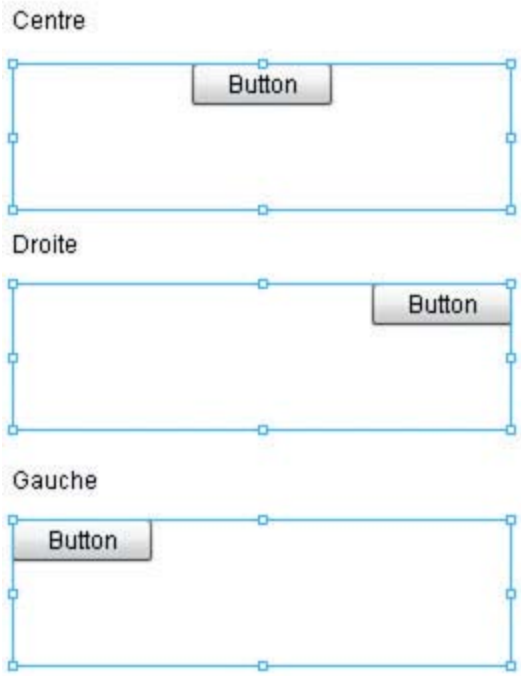
Dimensionner et positionner des composants
Positionner des composants

- Le positionnement automatique
  - Par défaut, Flex place les composants de façon automatique en fonction des règles de positionnement de leur conteneur
  - Exemple
    - un composant intégré dans le conteneur HGroup sera automatiquement positionné en haut à gauche, aux coordonnées x=0 et y=0
  - On peut néanmoins intervenir sur ce positionnement en modifiant certaines propriétés du conteneur
  - Le tableau suivant présente les propriétés les plus communément modifiées

Propriété	Rôle
`horizontalAlign`	Aligne les composants par rapport à l'axe horizontal du conteneur : `Center` : centre le composant. `Left` : place le composant à gauche. `Right` : place le composant à droite.
`verticalAlign`	Aligne les composants par rapport à l'axe vertical du conteneur : `Bottom` : place le composant en bas. `Middle` : place le composant au centre. `Top` : place le composant en haut.
`BasicLayout (Spark.layouts.BasicLayout)`	Format de présentation applicable aux composants conteneurs, qui permet de positionner leurs contenants à l'aide de leurs coordonnées X et Y.
`HorizontalLayout (Spark.layouts. HorizontalLayout)`	Format de présentation applicable aux composants conteneurs, qui permet de positionner leurs contenants horizontalement.
`VerticalLayout (Spark.layouts. VerticalLayout)`	Format de présentation applicable aux composants conteneurs, qui permet de positionner leurs contenants verticalement.
`TileLayout (Spark.layouts. TileLayout)`	Format de présentation applicable aux composants conteneurs, qui permet de positionner leurs contenants sous forme de tableaux.
`paddingBottom`	Marge inférieure du conteneur.
`paddingLeft`	Marge gauche du conteneur.
`paddingRight`	Marge droite du conteneur.
`paddingTop`	Marge supérieure du conteneur.

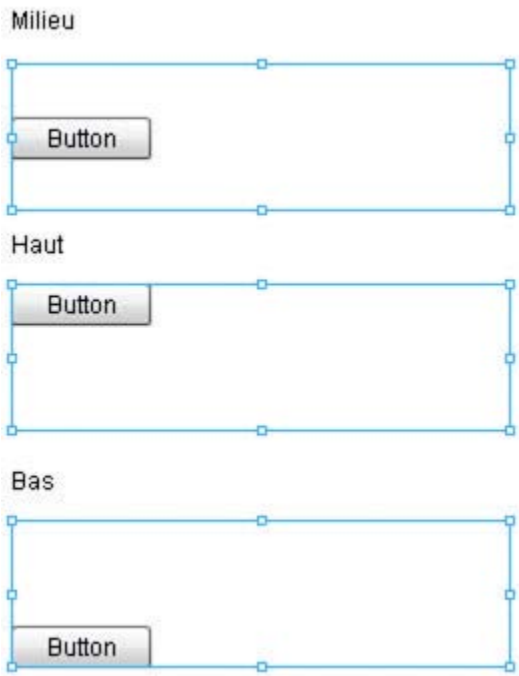
Exemple : *La propriété horizontalAlign*

```
<!-- CENTRE -->
<s:Label x=”10” y=”17” text=”Centre” width=”250”/>
<s:HGroup x=”10” y=”43” width=”250” height=”74”
    horizontalAlign=”center”>
<s:Button id=”btn1” label=”Button”/>
</s:HGroup>
<!-- DROITE -->
<s:Label x=”10” y=”129” text=”Droite” width=”250”/>
<s:HGroup x=”10” y=”153” width=”250” height=”74”
    horizontalAlign=”right”>
<s:Button id=”btn2” label=”Button”/>
</s:HGroup>
<!-- GAUCHE -->
<s:Label x=”10” y=”247” text=”Gauche” width=”250”/>
<s:HGroup x=”10” y=”271” width=”250” height=”74” horizontalAlign=”left”>
<s:Button id=”btn3” label=”Button”/>
</s:HGroup>
```



- *La propriété verticalAlign*

```
<!-- MILIEU -->
<s:Label x="10" y="17" text="Milieu" width="250"/>
<s:HGroup x="10" y="43" width="250" height="74"
    verticalAlign="middle">
<s:Button id="btn1" label="Button"/>
</s:HGroup>
<!-- HAUT -->
<s:Label x="10" y="129" text="Haut" width="250"/>
<s:HGroup x="10" y="153" width="250" height="74"
    verticalAlign="top">
<s:Button id="btn2" label="Button"/>
</s:HGroup>
<!-- BAS -->
<s:Label x="10" y="247" text="Bas" width="250"/>
<s:HGroup x="10" y="271" width="250" height="74"
    verticalAlign="bottom">
<s:Button id="btn3" label="Button"/>
</s:HGroup>
```

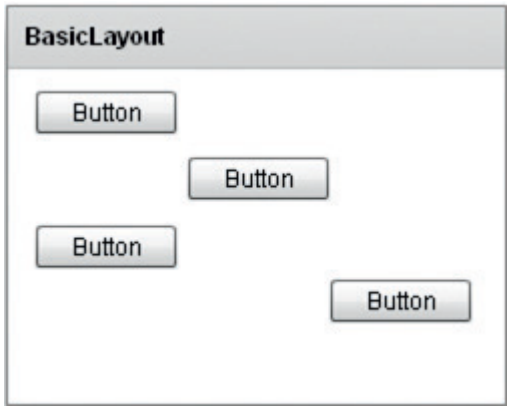


Dimensionner et positionner des composants
Positionner des composants

- Les formats de présentation (layouts)
  - La bibliothèque Spark du framework Flex developpe la notion de layouts :
    - des formats de présentation applicables aux composants conteneurs :
      - Panel, VGroup...
  - Ils permettent de positionner les composants de la bibliothèque librement (BasicLayout), horizontalement (HorizontalLayout), verticalement (VerticalLayout) ou en tableau (TileLayout)

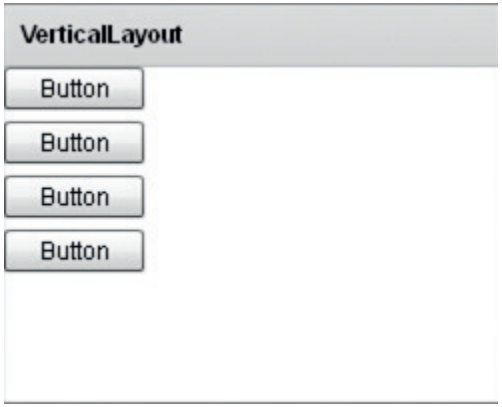
Exemples : 4 formats appliqués à un Panel

```
<!-- BasicLayout -->
<s:Panel x=”31” y=”35” width=”250” height=”200” title=”BasicLayout”>
<s:layout>
<s:BasicLayout/>
</s:layout>
<s:Button x=”14” y=”11” label=”Button” id=”btn1”/>
<s:Button x=”90” y=”44” label=”Button” id=”btn2”/>
<s:Button x=”14” y=”78” label=”Button” id=”btn3”/>
<s:Button x=”161” y=”105” label=”Button” id=”btn4”/>
</s:Panel>
```

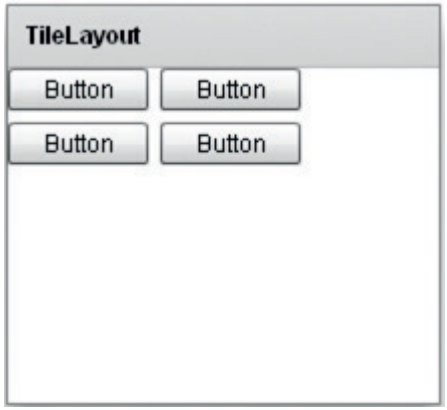



```
<!-- HorizontalLayout -->
<s:Panel x=”318” y=”35” width=”320” height=”200”
    title=”HorizontalLayout”>
<s:layout>
<s:HorizontalLayout/>
</s:layout>
<s:Button label=”Button” id=”btn5”/>
<s:Button label=”Button” id=”btn6”/>
<s:Button label=”Button” id=”btn7”/>
<s:Button label=”Button” id=”btn8”/>
</s:Panel>
```

```
<!-- VerticalLayout -->
<s:Panel x="32" y="289" width="250" height="200" title="VerticalLayout">
<s:layout>
<s:VerticalLayout/>
</s:layout>
<s:Button label="Button" id="btn9"/>
<s:Button label="Button" id="btn10"/>
<s:Button label="Button" id="btn11"/>
<s:Button label="Button" id="btn12"/>
</s:Panel>
```



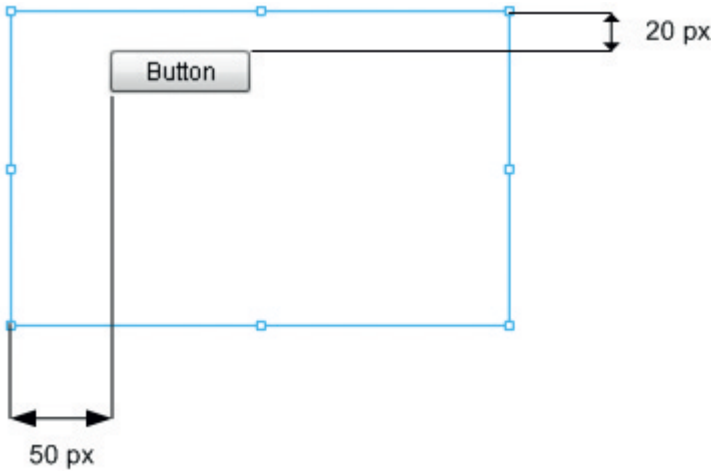

```
<!-- TileLayout -->
<s:Panel x="318" y="289" width="217" height="200" title="TileLayout">
<s:layout>
<s:TileLayout/>
</s:layout>
<s:Button label="Button" id="btn13"/>
<s:Button label="Button" id="btn14"/>
<s:Button label="Button" id="btn15"/>
<s:Button label="Button" id="btn16"/>
</s:Panel>
```



La propriété padding

- L’exemple de code suivant illustre le positionnement des composants à l’aide des marges

```
<s:HGroup x=”10” y=”10” width=”250” height=”158”
paddingLeft=”50” paddingTop=”20”>
<s:Button id=”btn” label=”Button”/>
</s:HGroup>
```



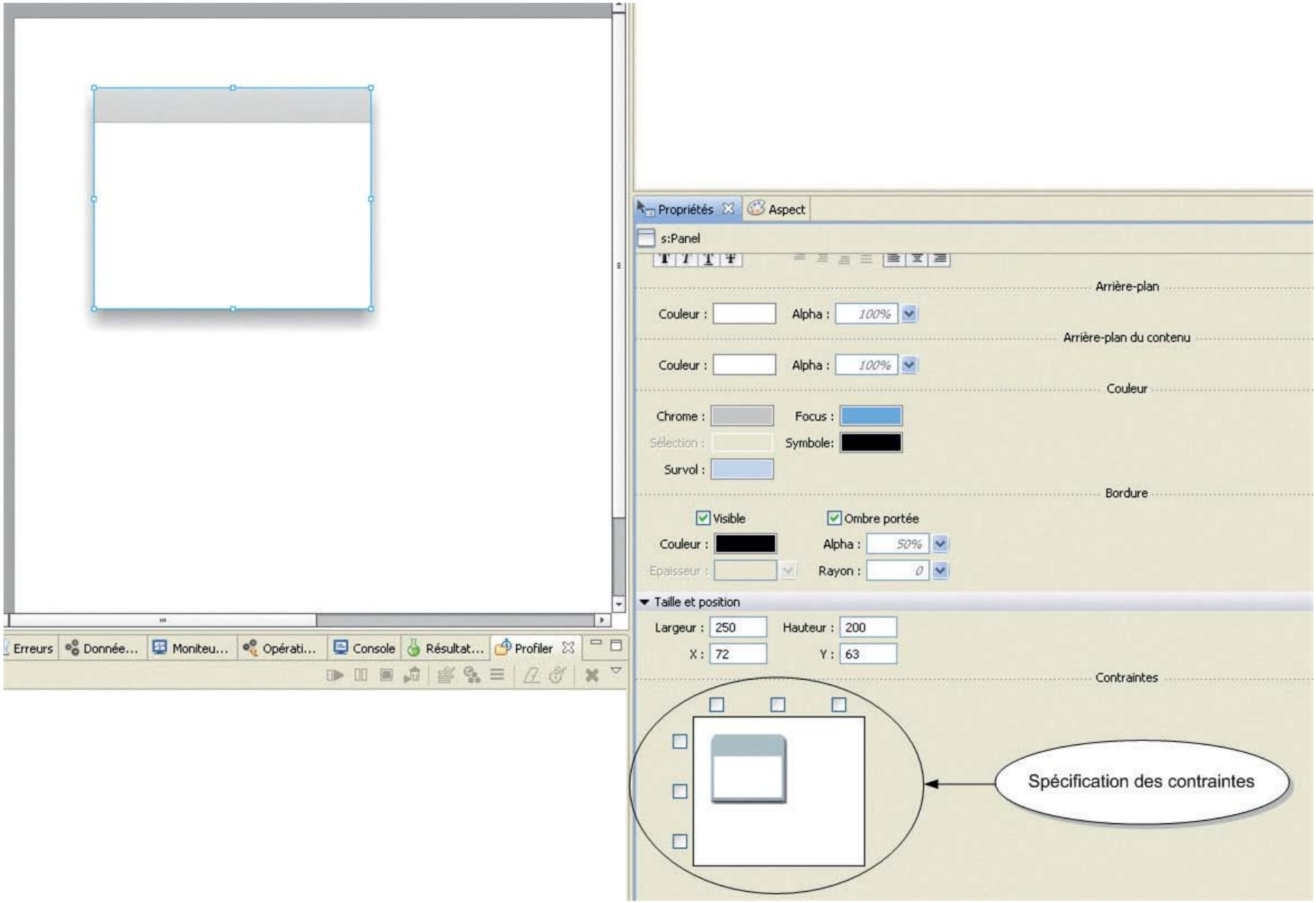
Dimensionner et positionner des composants

- Le positionnement à l'aide de contraintes
  - appelé positionnement par ancrage
  - Il peut s'effectuer de deux façons :
    - En modifiant les propriétés de positionnement par ancrage
    - En utilisant les propriétés de design en bas à droite du framework. Ces propriétés apparaissent quand on clique sur l'objet dessiné

Dimensionner et positionner des composants

- Propriétés de positionnement par ancrage

Propriété	Rôle
`horizontalCenter`	Distance exprimée en pixels entre le centre de l'axe horizontal du composant et celui de son conteneur.
`verticalCenter`	Distance exprimée en pixels entre le centre de l'axe vertical du composant et celui de son conteneur.
`top`	Distance exprimée en pixels entre le bord supérieur du composant et celui de son conteneur.
`bottom`	Distance exprimée en pixels entre le bord inférieur du composant et celui de son conteneur.
`left`	Distance exprimée en pixels entre le bord gauche du composant et celui de son conteneur.
`right`	Distance exprimée en pixels entre le bord droit du composant et celui de son conteneur.

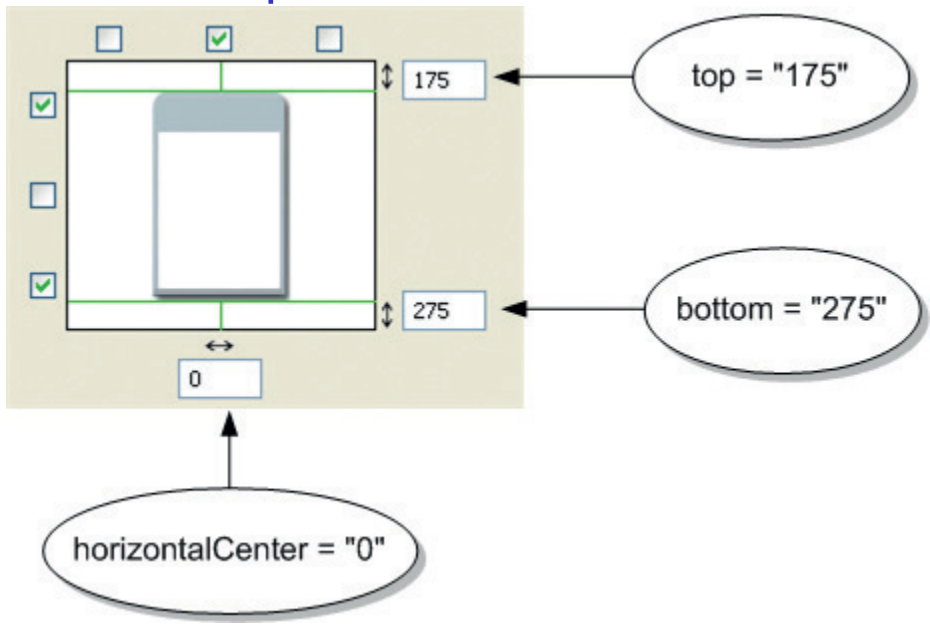

Propriétés
Aspect
s:Panel
Arrière-plan
Couleur :
Alpha :
100%
Arrière-plan du contenu
Couleur :
Alpha :
100%
Couleur
Chrome :
Focus :
Sélection :
Symbole:
Survol :
Bordure
Visible
Ombre portée
Couleur :
Alpha :
50%
Epaisseur :
Rayon :
0
Taille et position
Largeur :
250
Hauteur :
200
X :
72
Y :
63
Contraintes
Spécification des contraintes
Erreurs
Donnée...
Moniteu...
Opérati...
Console
Résultat...
Profiler

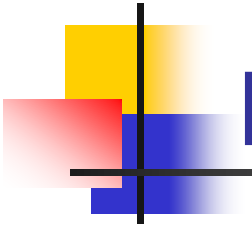
Exemple d'application

- Soit une application servant de conteneur à un composant Panel correspondant au code suivant :

```
<s:Panel x="72" y="63" width="250" height="200"
    horizontalCenter="0" top="175" bottom="275" >
</s:Panel>
```

- Les contraintes de positionnement de Panel sont :
  - doit se situer au centre de l'application : horizontalCenter="0" ;
  - doit se situer à 175 pixels du bord supérieur de l'application : top="175" ;
  - doit se situer à 275 pixels du bord inférieur de l'application : bottom="275".





Réaliser le design des interfaces

- Les styles
  - Le style proposé par défaut dans une application Flex peut parfaitement convenir
  - Flex permet cependant de personnaliser le style d'une application au moyen du langage CSS (*Cascading Style Sheet)* en agissant sur les propriétés
    - Color, fontFamily, fontSize, fontStyle, fontWeight, paddingBottom, paddingLeft, paddingRight, paddingTop, textAlign, textDecoration, textIndent
  - Pour visualiser les propriétés de style d'un composant, il suffit de le sélectionner, cliquer sur le bouton Vue par catégories

Réaliser le design des interfaces

Les styles

- La balise Style : style1.mxml
  - consiste à utiliser la balise <fx:Style>

```
<fx:Style>
    @namespace s “library://ns.adobe.com/flex/spark”;
    @namespace mx “library://ns.adobe.com/flex/mx”;
    s|Label{
            fontSize: 10;
            color:red;
    }
    s|Button {
            fontSize: 20;
    }
</fx:Style>
```

Réaliser le design des interfaces

Les styles

- **Le fichier CSS externe**
  - La seconde méthode d'implémentation consiste à faire appel à un fichier CSS externe
  - Pour ce faire, nous utiliserons de nouveau la balise <fx:Style> mais en lui précisant le chemin d'accès du fichier CSS à utiliser

```
...
<fx:Style source="feuilleDeStyleCSS.css">
...
```

  - Le fichier CSS peut être édité à l'aide de la perspective de développement en choisissant la création d'un nouveau fichier CSS

Réaliser le design des interfaces

Les styles

- Utilisation des images
  - Pour utiliser des images, il convient de faire appel à la méthode Embed()
  - Ici : l'image monImage.jpg est employée comme arrière-plan

    backgroundImage : Embed('monImage.jpg')

Réaliser le design des interfaces

Les styles

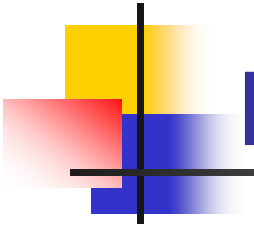
- CSS et ActionScript
  - La troisième méthode d'implémentation consiste à créer le style CSS à l'aide du langage ActionScript
  - Ceci est possible grâce à la classe StyleManager qui permet de définir de nouveaux styles et de les appliquer aux composants de l'application

Exemple

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="absolute"  creationComplete="appliquerStyle()" >
<mx:Script>
<![CDATA[
// Importation de la classe StyleManager
import mx.styles.StyleManager;
// Définition du style
private var styleActionScript:CSSStyleDeclaration;
private function appliquerStyle():void{
// Instanciation de la classe CSSStyleDeclaration;
styleActionScript = new CSSStyleDeclaration('styleActionScript');
// Création des propriétés
styleActionScript.setStyle('fontSize',20);
// Affectation des propriétés à la classe Label
StyleManager.setStyleDeclaration("Label",styleActionScript,true);
}]>
</mx:Script>
<mx:Label id="lab" x="10" y="10" text="Étiquette"/>
</mx:Application>
```

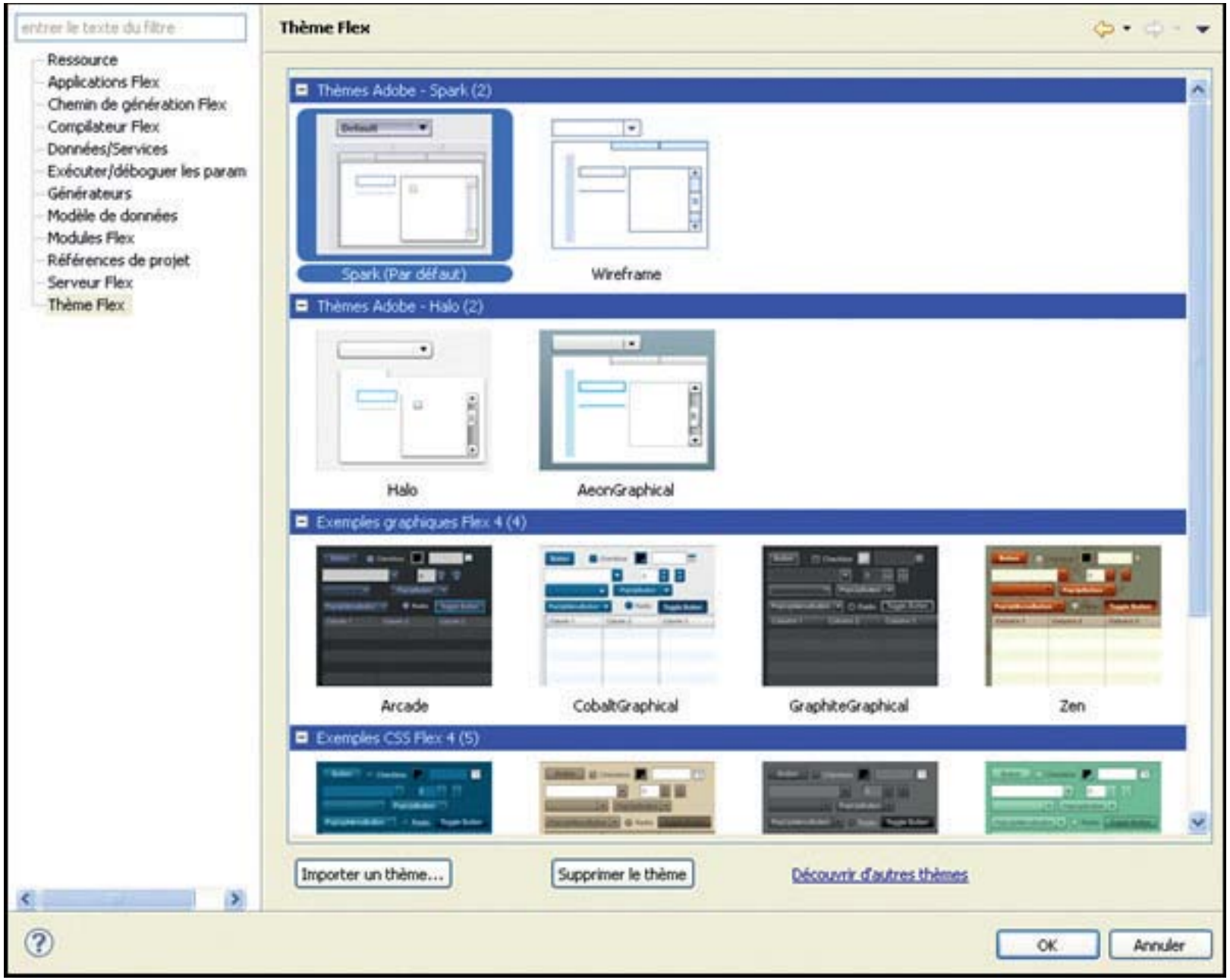
Commentaires

- Instancier la classe CSSStyleDeclaration permettant de déclarer un nouveau style.
- Ajouter ensuite des propriétés CSS grâce à la méthode setStyle()
- Enfin, affecter ce style à la classe Label en employant la méthode setStyleDeclaration(*nom de la classe CSS, style, mise à jour (true/false))*
- *À noter que l'ensemble de ces instructions* est contenu dans la fonction appliquerStyle(), exécutée à la fin de la création de l'application.
- Grâce à la classe StyleManager, il devient possible de manipuler le style CSS de notre application
- Ceci ouvre la voie vers une multitude de possibilités, notamment celle du dynamisme applicatif, c'est-à-dire le changement de style en fonction de l'utilisateur connecté ou lors d'un déclenchement d'événement



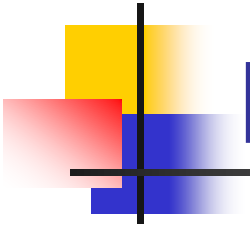
Réaliser le design des interfaces

- Les thèmes
  - La grande nouveauté du framework Flex 4 est l'utilisation de thèmes applicatifs
    - Un thème est un ensemble de fichiers .CSS et ActionScript réalisant le design de tout ou partie des composants de l'application (boutons, panneaux, polices…) et gérant leur comportement
  - Pour modifier le thème de l'application, il suffit
    - d'effectuer un clic droit sur le nom du projet, de choisir l'option Propriété et de se positionner sur l'onglet Thème Flex


entrer le texte du filtre
Ressource
Applications Flex
Chemin de génération Flex
Compilateur Flex
Données/Services
Exécuter/déboguer les param
Générateurs
Modèle de données
Modules Flex
Références de projet
Serveur Flex
Thème Flex
Thème Flex
Thèmes Adobe - Spark (2)
Spark (Par défaut)
Wireframe
Thèmes Adobe - Halo (2)
Halo
AeonGraphical
Exemples graphiques Flex 4 (4)
Arcade
CobaltGraphical
GraphiteGraphical
Zen
Exemples CSS Flex 4 (5)
Importer un thème...
Supprimer le thème
Découvrir d'autres thèmes
OK
Annuler

Flash Builder

La bibliothèque graphique FXG

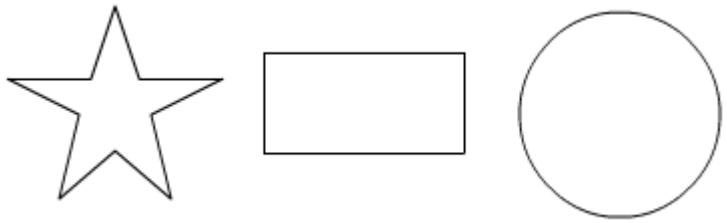


FXG

- Définition
  - permet de créer des éléments graphiques vectoriels dit «primitifs»
    - Le rectangle <s:Rect>
    - l'ellipse <s:Ellipse>
    - et le chemin de points <s:Path> (ensemble
    - de points qui, reliés, forment une droite)
  - En combinant ces éléments basiques, on crée des figures complexes
  - On peut leur affecter
    - Couleur, Luminosité, axes 3D…

Exemple : fxg1.mxml

```
<!– réalisation d'une étoile -->
<s:Path data="M 55 0 L 67 36 L 109 36 L 73 54
    L 83 96 L 55 72 L 27 96 L 37 54
    L 1 36 L 43 36 Z" x="25" y="11">
    <!-- Tracé -->
    <s:stroke>
      <s:SolidColorStroke color="black"/>
    </s:stroke>
</s:Path>
<!-- Réalisation d'un rectangle -->
<s:Rect width="100" height="50" x="154" y="34">
    <s:stroke>
      <s:SolidColorStroke color="black"/>
    </s:stroke>
</s:Rect>
<!-- Realisation d'une ellipse -->
<s:Ellipse width="100" height="102" x="282" y="14">
    <s:stroke>
      <s:SolidColorStroke color="black"/>
    </s:stroke>
</s:Ellipse>
</s:Application>
```

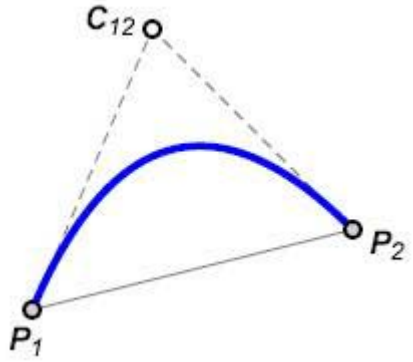


Les mouvements

Instruction	Action	Exemple
M ou m (Move)	Déplace le curseur du pinceau selon les coordonnées X et Y :	M 10 20 Déplace la ligne sur 10, 20 pixels.
L ou l (Ligne)	Trace une ligne à partir de deux coordonnées X et Y fournies en paramètre :	L 50 30 Trace une ligne sur 50, 30 pixels.
H ou h	Trace une ligne horizontale sur une longueur en pixel donnée	H 40 Trace une ligne horizontale sur 40 pixels.
V ou v (ligne verticale)	Trace une ligne verticale sur une longueur en pixel donnée	V 100 Ligne verticale sur 100 pixels.
Q ou q	Trace des courbes quadratique de Bézier	Q 110 45 90 30 Trace une courbe sur 90, 30 avec le point de contrôle sur 110, 45.
C ou c	Trace des courbes cubique de Bézier :	C 45 50 20 30 10 20 Trace une courbe sur 10, 20 avec le premier point de contrôle sur 45, 50 et le second point de contrôle sur 20, 30.
Z ou z	Ferme le chemin.	

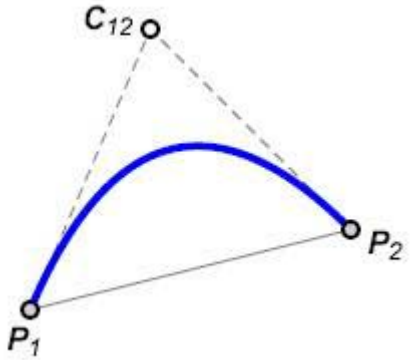
Les mouvements

- Courbe de bézier quadratique
  - On part d'un segment P1P2, et on ajoute un troisième point C12 en dehors de ce segment
  - La courbe de Bézier passe par les points de contrôle P1 et P2, et approche seulement le point C12
  - C'est comme si C12 exerce une certaine attraction sur le segment et le déforme en une courbe régulière

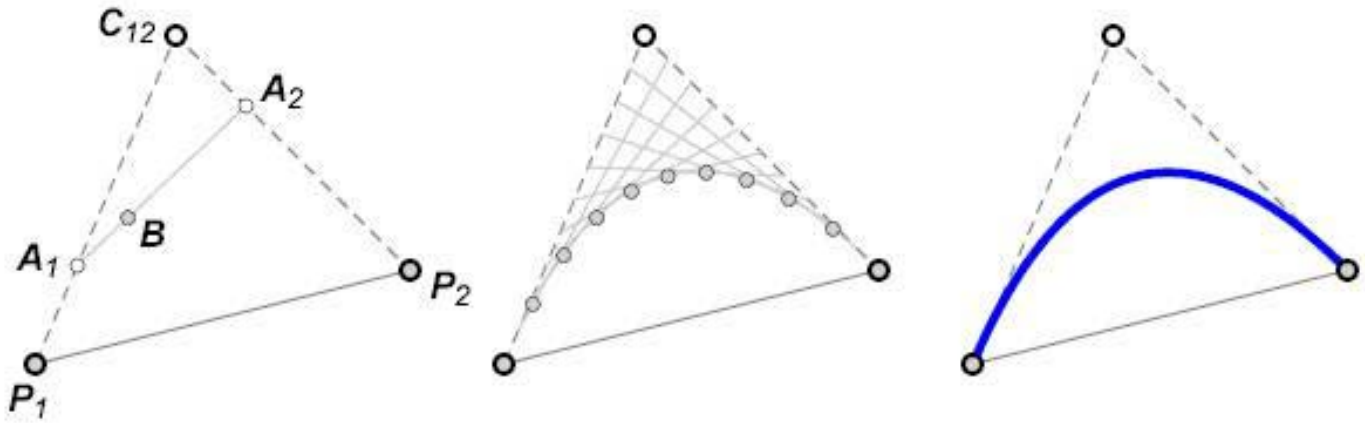


Les mouvements

- Syntaxe : Q|q x1, y1, x, y
  - Où :
    - x1, y1 : coordonnées du point de contrôle C12
    - x, y : coordonnées de P2, fin de tracé
    - P1 est supposé être le point précédent du tracé à partir duquel on trace la courbe de Bézier
  - Exemple :
    - <path fill="white" stroke="black" stroke-width="2" d="M 50,100 Q100,50,125,100" />



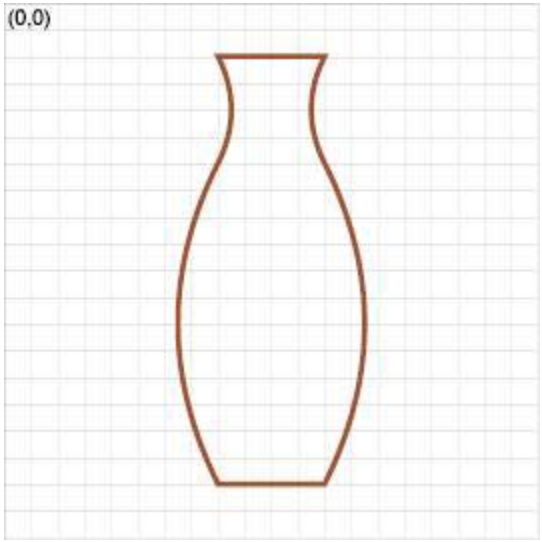
- Algorithme de construction et intérêt de la représentation graphique
  - On place un point A1 à 30% du segment P1 C12 ainsi que A2 à 30% de C12 P2
  - On joint ces points et place B à 30% du segment A1 A2 obtenu
  - Ce point B appartient à la courbe de Bézier
  - En répétant cette construction en changeant le rapport de 0% à 100% avec un pas de 10%, on obtient la figure centrale
  - Tous les segments construits sont des tangentes à la courbe - ils constituent l'enveloppe convexe de la courbe - et tous ces points B appartiennent à la courbe de Bézier qui n'est dans ce cas qu'une simple parabole

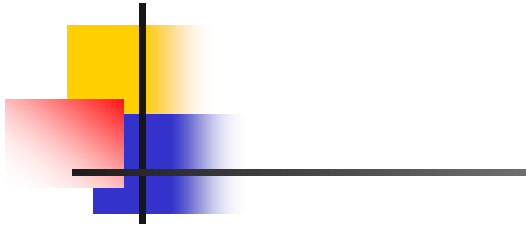


Les mouvements

- Courbe de Bézier quadratique
  - Exemple : bezier-quadratique.svg

```
<s Path Data="M  80,180
              Q  50,120  80,60
              Q  90, 40  80,20
              Q 100, 20 120,20
              Q 110, 40 120,60
              Q 150,120 120,180
              Z" />
```





Flash Builder 4

L’habillage (skinning) des composants Flex 4

L’habillage (skinning) des composants Flex 4

- Principe
  - Un des principaux changements de Flex 4 est la séparation de l'apparence (Skin) par rapport au comportement d'un composant
  - Avec Flex 3, la partie Skinning d'une application se résumait bien souvent à **modifier les styles de composants de base** (styles CSS)
  - Grâce au format d’échange FXG, il est possible de faire des habillages personnalisés intéressants

L'habillage (skinning) des composants Flex 4

- Créer un habillage ou skin personnalisé
  - Avant d'aller plus loin, définissons la notion d'habillage.
  - Définition
    - Un habillage (*skin) est un fichier MXML annexe à l'application* principale, comportant l'ensemble des éléments définissant les propriétés graphiques d'un composant
    - Il nous est alors possible de définir la couleur, la police, les contours et tout autre élément graphique supporté par le composant destinataire de l'habillage
    - Ainsi, on peut définir un comportement graphique sur un bouton lorsque celui-ci est affiché à l'écran puis un autre lorsque l'utilisateur clique sur ce dernier

L’habillage (skinning) des composants Flex 4

- On se lance tout doucement
  - Commencer par un bouton le plus simple possible,
    - c'est-à-dire un rectangle arrondi avec un label au dessus
    - Pour le rectangle arrondi, on va prendre la primitive Spark «Rect» et pour le label, un composant «Label» Spark

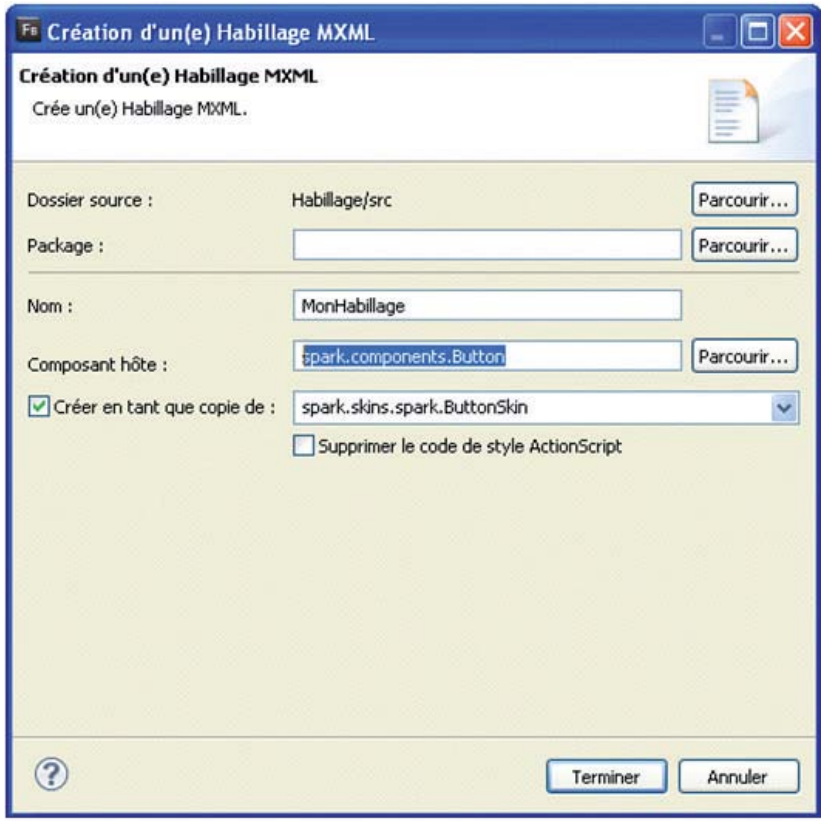
Skin1.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark">
<s:Group verticalCenter="0" horizontalCenter="0">
    <s:Rect id="rect" radiusX="4" radiusY="4" top="0“ right="0" bottom="0" left="0">
    <!-- Couleur de fond -->
        <s:fill><s:SolidColor color="0x0099FF" /></s:fill>
        <!-- Couleur de la bordure -->
        <s:stroke>
            <s:SolidColorStroke color="0x222222" weight="2"/>
        </s:stroke>
    </s:Rect>
    <s:Label text="Bouton" color="0x222222" textAlign="center" verticalAlign="middle"
        horizontalCenter="0" verticalCenter="1" left="12" right="12" top="6" bottom="6"
    />
</s:Group>
</s:Application>
```

- Commentaires
  - Le composant créé est un ensemble d'éléments graphiques regroupés par la balise <s:Group>
    - Cette balise assure également le rendu graphique des éléments qui s'y trouvent
    - En son absence, rien ne s'affiche à l'écran
  - À l'intérieur de ce groupe, nous observons la présence du rectangle avec quelques paramètres supplémentaires qui colorent le fond et la bordure du composant
  - Notons également la présence du composant Label qui spécifie l'action associée au bouton
- Cependant, il ne s'agit ici que d'une production graphique
  - Il nous faut maintenant la transformer en un habillage pouvant être utilisé sur des composants Flex 4

Procédure

- Pour ce faire, grâce à un simple clic droit sur le nom du projet, choisir l'option Nouveau puis Habillage MXML
- Dans la fenêtre qui se présente à vous, saisir MonHabillage dans la zone Nom (correspondant au nom donné à notre habillage)
- Comme nous souhaitons habiller les boutons à l'aide de notre production graphique, saisissons spark.components. Button dans la zone de texte composant hôte
- Une fois ceci terminé, nous voyons dans l'arborescence de notre projet un nouveau fichier MonHabillage.mxml



- Procédure
  - Ce fichier est découpé en parties fonctionnelles qui définissent l'habillage du composant bouton
  - Les différents états de notre composant sont les suivants :
    - Up : état normal du bouton
    - Over : état du bouton lors du survol de la souris
    - Down : état du bouton lorsque l'utilisateur clique dessus
    - Disabled : état du bouton désactivé

- A faire : skin2.mxml
  - Examinez le fichier Mon Habillage.mxml et essayez de le comprendre

Exploiter l’habillage dans une l’application

- Notre habillage étant terminé, voyons à présent comment l’utiliser dans une application Flex
- Pour utiliser l’habillage que nous venons de créer sur un composant de type bouton, il suffit d’écrire cette ligne de code dans le fichier principal de l’application que vous avez créée en début de chapitre :

```
<s:Button skinClass=”MonHabillage” label=”Bouton habillé” />
```