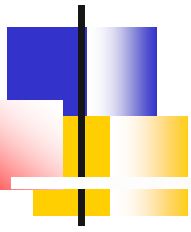


Flex 4

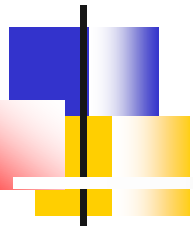
Accès aux services de données de Flex



Introduction

■ Objectif du chapitre

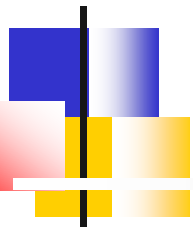
- Comprendre le fonctionnement des communications client/serveur dans une application Flex
- Interagir avec des données ou des services extérieurs pour alimenter vos **composants** et charger dynamiquement des **ressources**
- Interagir avec une **base de données**



Récupération des données

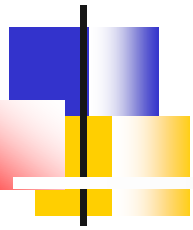
■ Comme vous le savez

- Flex ne peut accéder aux informations contenues dans une base de données en raison de son appartenance à la couche graphique, il va demander à un tiers, la couche métier, de réaliser ce travail, exactement comme nous demandons parfois à une personne de notre entourage de nous rendre un **service**
- Pour communiquer avec cette couche, Flex dispose de trois composants RPC (Remote Procedure Call) :
 - HTTPService, également dénommé REST-Style webservice
 - WebService
 - RemoteObject



Chargement dynamique

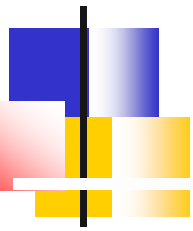
- Première application : chargement dynamique des données XML
 - Dans les chapitres précédents, les données manipulées étaient incluses dans les applications
 - soit directement dans le fichier MXML
 - soit dans un fichier séparé en utilisant la propriété **source** du composant `<mx:XML>`
 - Dans ces deux cas, les données sont chargées à la compilation dans le fichier SWF généré, ce qui implique une recompilation à chaque modification des données
 - D'où l'idée du **chargement dynamique** :
 - Le fichier ne sera chargé qu'à l'exécution et non à la compilation
 - Un ralentissement sera perçu au premier chargement, mais après, le fichier sera stocké en cache dans le navigateur



Chargement dynamique

■ Les classes invoquant des URL

- L'URLRequest
 - Permet de récupérer les données rangées dans un fichier connu par son URL, à l'aide d'une **requête**
 - L'information est passée à la classe URLLoader
- La classe URLLoader
 - Gère les événements concernant la progression dans le chargement
 - ProgressEvent.PROGRESS...**
 - Event.Complete**
 - La donnée n'est récupérée que quand l'événement de chargement complet s'est produit



Récupération des données

■ Exemple : urlLoaderRequest1.mxml

- Soit à charger le fichier XML : `test.xml`
- On crée une application avec un conteneur qui va recevoir ce fichier au click d'un bouton (côté dynamique du chargement)

```
<mx:XML id="sourceCode"/>
```

```
<mx:Panel>
```

```
    <mx:TextArea width="300" height="300" text="{sourceCode}"/>
```

```
    <mx:Button label="Afficher le code source" click="loadXML()"/>
```

```
</mx:Panel>
```

- La fonction `loadXML` va utiliser les classes `URLRequest` et `URLLoader` pour charger le fichier et le ranger dans la variable `sourceCode` liée ici avec la TextArea

– La fonction loadXML :

//Déclaration du chargeur : loader

```
public var loader:URLLoader = new URLLoader ();
```

```
public function loadXML():void{
```

```
    //On déclare l'url du fichier
```

```
    var url:String="test.xml";
```

```
    //On invoque la classe URLRequest avec l'url dans le constructeur
```

```
    var request:URLRequest = new URLRequest(url);
```

```
    //On ajoute un listener au loader onLoad : quand le chargement sera complet, on appelle onLoad
```

```
    loader.addListener(Event.COMPLETE, onLoad);
```

```
    //On demande au loader de charger les données pointées par URLRequest
```

```
    loader.load(request);
```

```
}
```

```
public function onLoad(evt:Event):void{
```

```
    //L'écouteur traite l'événement « chargement complet » en associant la donnée XML à la variable sourceCode
```

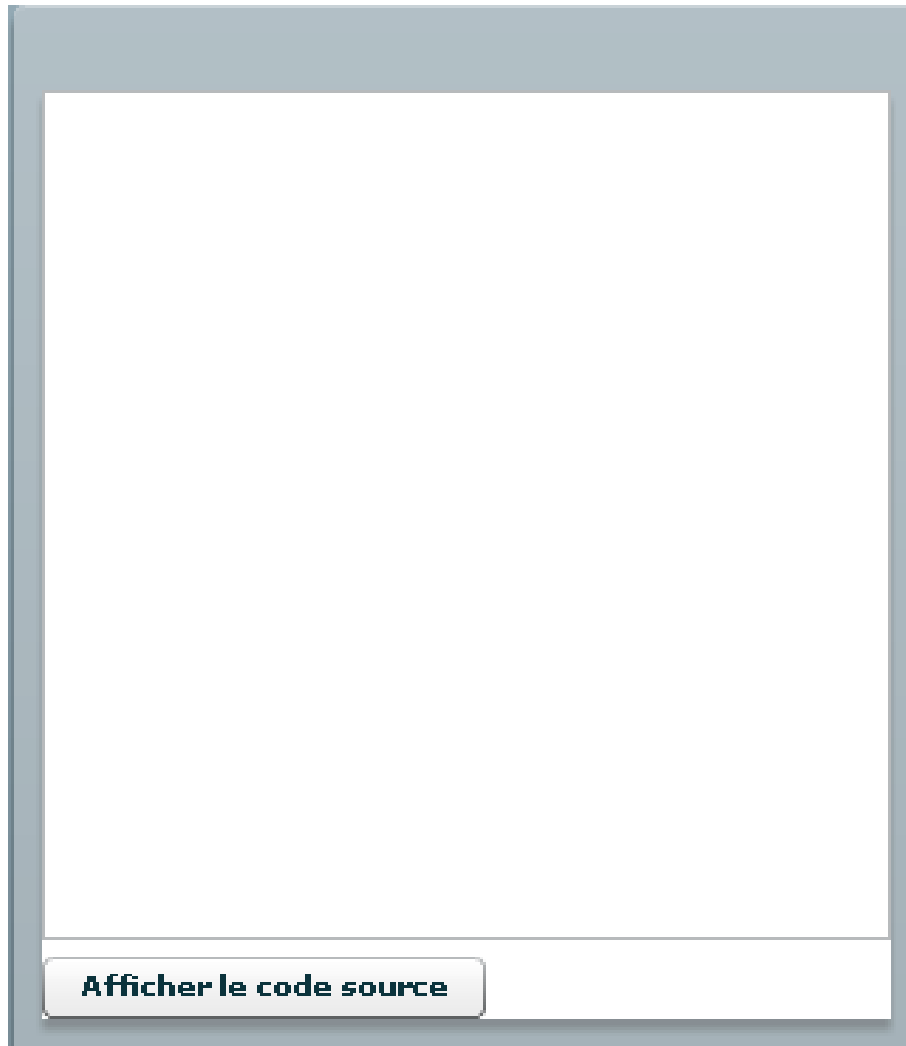
```
        sourceCode=new XML(loader.data);
```

```
}
```

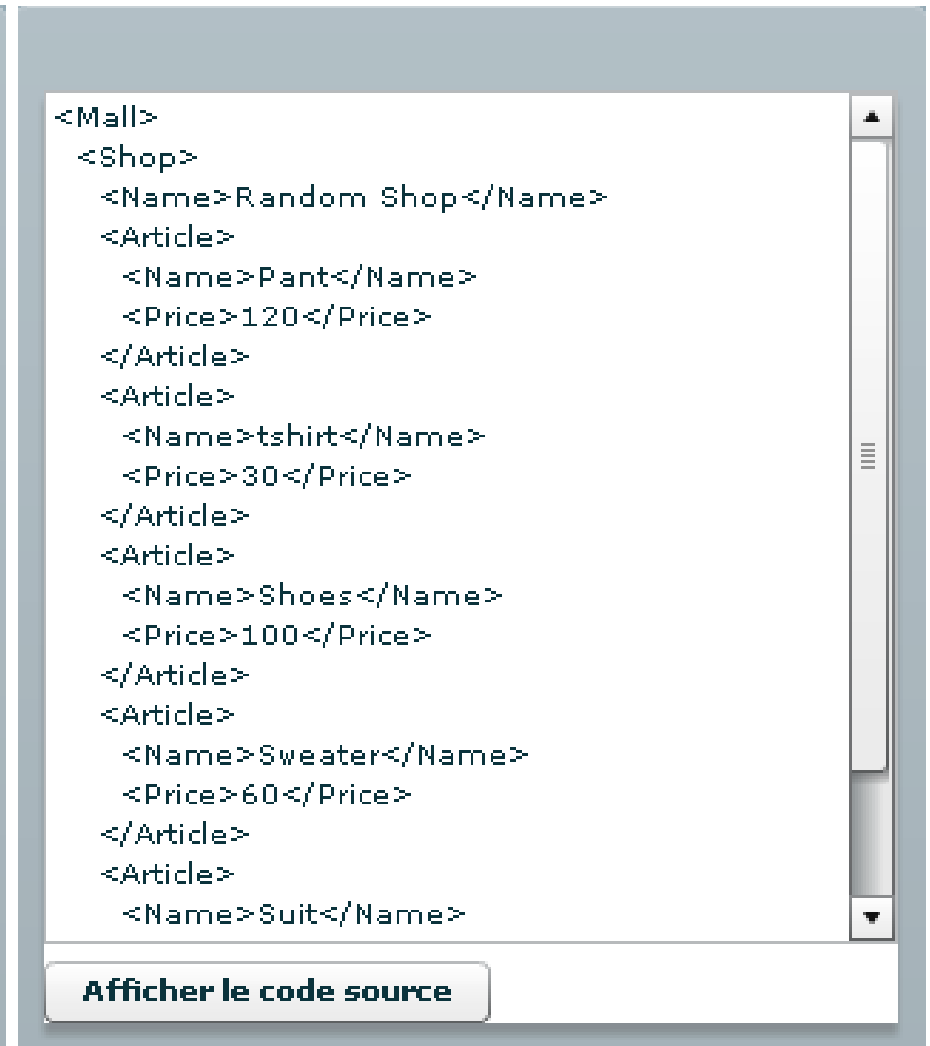
■ Le code complet

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="vertical">
  <mx:XML id="sourceCode"/>
  <mx:Panel>
    <mx:TextArea width="300" height="300" text="{sourceCode}"/>
    <mx:Button label="Afficher le code source" click="loadXML()"/>
  </mx:Panel>
  <mx:Script>
  <![CDATA[
    public var loader:URLLoader = new URLLoader ();
    public function loadXML():void{
      var url:String="test.xml";
      var request:URLRequest = new URLRequest(url);
      loader.addEventListener(Event.COMPLETE, onLoad);
      loader.load(request);
    }
    public function onLoad(evt:Event):void{
      sourceCode=new XML(loader.data);
    }
  ]]>
  </mx:Script>
</mx:Application>
```

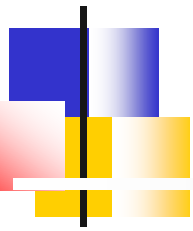

Résultat : urlLoaderRequest1.mxml



Avant d'appuyer sur le bouton



Après

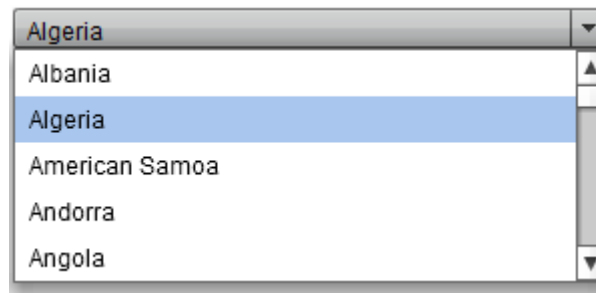


Récupération des données

■ Un autre exemple : urlLoaderRequest2.mxml

- Ici, le fichier XML est distant
- Vous pouvez d'abord le consulter directement avant de l'utiliser
- Il représente des pays...

<http://www.solayaresorts.com/xml/countries.xml?nocache=143709>



```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" horizontalAlign="center"
  initialize="initializeHandler(event)" verticalAlign="middle">
```

```
  private var loader:URLLoader;
```

```
  private function initializeHandler(event:Event):void{
```

```
    loader= new URLLoader();
```

```
    loader.addEventListener(Event.COMPLETE, countriesCompleteHandler);
```

```
    //demande de chargement des données
```

```
    var
```

```
    url:String="http://www.solayaresorts.com/xml/countries.xml?nocache=143709";
```

```
    loader.load(new URLRequest(url));
```

```
  }
```

```
  private function countriesCompleteHandler(event:Event):void{
```

```
    //Une fois les données chargées, on les récupère dans xml
```

```
    var xml:XML = new XML(loader.data);
```

```
    //On lie les données au conteneur par dataProvider
```

```
    country.dataProvider = xml.children();
```

```
  }
```

```
]]>
```

```
</mx:Script>
```

```
<mx:ComboBox id="country"/>
```



Accès aux données distantes

■ Deuxième application : invocation de services distants

- Il d'agit de récupérer des données mises à jour par ces services : météo, informations touristiques, e_commerce...
- Il existe plusieurs classes qui font partie des composants d'une procédure d'appel à distance (RPC) dans une architecture client/serveur :
 - **HttpService**
 - Appel par des requêtes GET ou POST
 - **WebService**
 - Appel à des procédures sur des services Web par les protocoles HTTP ou SOAP
 - **RemoteObject**
 - Appel à des procédures sur des objets Java distants
- Un composant RPC, côté client, que vous pouvez créer en MXML ou ActionScript, appelle un service distant, puis stocke les données de réponse du service dans un objet ActionScript à partir duquel vous pouvez facilement obtenir les données

Côté Client

Tiers de présentation

Services RPC

- HTTPService : communication par des requêtes GET ou POST
- HTTPWebService : appel à des procédures sur des services Web par les protocoles HTTP ou SOAP
- RemoteObject : appel à des procédures sur des objets Java distants

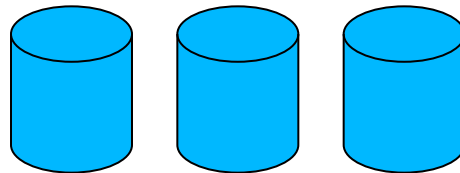
Requête

Réponse

Côté Serveur

Implémentation d'un service RPC :
web services, http ou objets Java

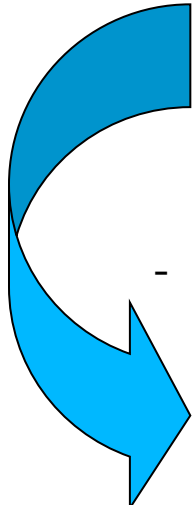
Ressources externes/Base de données

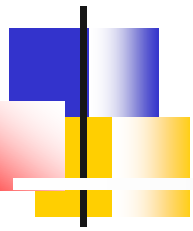




Accès aux données distantes

■ HTTPService

- 
- Classe ActionScript permettant de créer des objets permettant de **faire des requêtes** vers de simples services HTTP
 - tels que des fichiers texte, des fichiers XML ou des scripts/pages comme URLRequest
 - S'utilise à travers la balise **<mx:HTTPService>** ou un objet AS
 - Parmi ses attributs, on trouve :
 - **url**=emplacement du fichier recherché
 - **method**="GET|POST|..."
 - **resultFormat**="object|array|xml|e4x|flashvars|text"
 - Parmi ses propriétés, on trouve :
 - **lastResult** : //contient les données récupérées
 - **id** : l'identifiant qui sera utilisé pour faire référence à l'objet HTTPService créé dans l'application



Accès aux données distantes

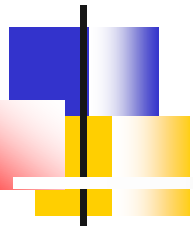
Parmi ses méthodes, on trouve :

- `send()` : // qui exécute la requête `HTTPService`
 - En effet, créer un objet `HTTPService` ne fait pas automatiquement la requête pour charger la donnée
 - Pour faire cette requête, on doit appeler la méthode `send()` en réponse à n'importe quel événement de l'application ou de l'utilisateur
- Par exemple
- Si vous voulez faire une requête aussi tôt que l'application démarre, vous pouvez appeler `send()` en réponse à l'événement `"initialize"`

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="initializeHandler(event)">
```

```
...
```

```
private function initializeHandler(event:Event):void{
    textService.send();
}
```



Accès aux données distantes

■ HTTPService : utilisation simple

- Le code suivant utilise du MXML pour créer un objet HTTPService qui charge du texte depuis un fichier .txt appelé **data.txt**, sauvegardé dans le même dossier que le fichier SWF compilé :

```
<mx:HTTPService id="textService" url="data.txt"/>
```




Accès aux données distantes

■ Exploitation de l'événement : **result**

- La méthode **send()** fait la requête, mais la réponse n'est retournée qu'une fois que l'événement concernant l'arrivée du résultat est produit
- Dans l'exemple suivant, l'événement "**result**" se produit quand une réponse complète a été retournée
 - L'exemple suivant affiche une Alert lorsque la donnée est chargée :

```
<mx:HTTPService id="textService" url="data.txt"  
    result="mx.controls.Alert.show('Données chargées')"  
>
```

- Dans l'exemple suivant, on utilise la propriété de l'objet HTTPService :
"lastResult"
 - Attention, cette propriété est typée comme Object
 - On utilise un cast String pour l'obtenir sous la forme d'une chaîne de caractères
 - L'exemple charge un fichier texte et l'affiche dans un TextArea :
httpService.mxml

```
<?xml ...initialize="initializeHandler(event)">
<mx:Script>
<![CDATA[
    private function initializeHandler(event:Event):void{
        textService.send();
    }
    private function resultHandler(event:Event):void{
        textArea.text = String(textService.lastResult);
    }
]]>
</mx:Script>
```

```
<mx:HTTPService id="textService" url="data.txt"  
result="resultHandler(event)"/>
```

```
<mx:TextArea id="textArea"/>  
</mx:Application>
```

- L'exemple suivant fait la même chose mais en utilisant le data binding : `httpService0.mxml`

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="initializeHandler(event)">
<mx:Script>
<![CDATA[

private function initializeHandler(event:Event):void{
textService.send();
}
]]>
</mx:Script>

<mx:HTTPService id="textService" url="data.txt"/>

<mx:TextArea id="textArea" text="{textService.lastResult}"/>
</mx:Application>
```

- Lorsque c'est possible, HTTPService désérialise la donnée qu'il charge, un peu comme s'il interprétait la donnée placée dans un tag <mx:Model>
- Par exemple, on considère la donnée XML suivante :
[asests/countries.xml](#)

```
<countries>  
  <country>Sélectionner</country>  
  <country>Canada</country>  
  <country>USA</country>  
</countries>
```

- Si vous chargez cette data en utilisant HTTPService, elle sera parsée en un objet nommé "countries" qui contient un tableau nommé "country", chaque élément correspondant à un élément <country>

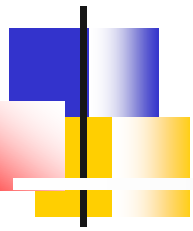
- Exemple : `httpService1.mxml`

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
verticalAlign="middle" horizontalAlign="center"
initialize="initializeHandler(event)">
<mx:Script>
<![CDATA[

private function initializeHandler(event:Event):void{
countryService.send();
}
]]>
</mx:Script>

<mx:HTTPService id="countryService" url="countries.xml"/>

<mx:VBox>
<mx:ComboBox id="country"
  dataProvider="{countryService.lastResult.countries.country}"/>
</mx:VBox>
</mx:Application>
```



HTTPService

■ Type de retour d'une requête http

- Il est possible de choisir le type de retour de la réponse http grâce à la propriété **resultFormat** de la classe HTTPService qui accepte les valeurs suivantes :
 - **text** : résultat stocké directement dans une chaîne sans aucun traitement (c'est ce qu'on a vu précédemment)
 - **array** (tableau d'objets représentant l'ensemble des nœuds d'un même niveau dans l'arbre XML)
 - **e4x** : résultat renvoyé sous la forme d'une instance XML
 - **flashvars** (valeurs stockées sous forme de texte de type nom=valeur séparées par &)

■ Exemple : httpService2.mxml

//La requête

```
<mx:HTTPService url="../../assets/contenu.xml" id="donneesSite"
  resultFormat="e4x"/>
```

//Récupération du résultat en fonction du bouton appuyé

```
<mx:VBox>
  <mx:Label fontSize="24" id="enTete" text="Cliquez sur un bouton"/>
  <mx:Label id="corps" text="A vous de choisir !"/>
  <mx:HBox>
    <mx:Button label="Page d'accueil" click="afficherPage('accueil')"/>
    <mx:Button label="Page a propos" click="afficherPage('apropos')"/>
    <mx:Button label="Page de contact" click="afficherPage('contact')"/>
  </mx:HBox>
</mx:VBox>
```


//Traitement de la requête

```
import mx.rpc.events.resultEvent;
```

De l'EcmaScript rendu possible par le format e4x

// Contient le résultat courant du fichier XML

```
private var contenuSite:XMLList;
```

```
private function afficherPage(pageCible:String):void{
```

// Recherche du titre de la page cible

```
contenuSite = donneesSite.lastResult.page.(@id==pageCible).titre;
```

// Change le texte de l'étiquette

```
enTete.text = contenuSite;
```

// Recherche du corps de la page cible

```
contenuSite = donneesSite.lastResult.page.(@id==pageCible).texte;
```

```
corps.text = contenuSite;
```

```
}
```



Accès aux données distantes

■ Lancer une requête HTTP à un service distant

- Nous allons maintenant recourir au MXML pour demander de l'information à un service distant
- Exemple
 - Nous allons utiliser le service **localSearch** de l'API Yahoo pour rechercher un restaurant de San Francisco
 - Nous rangerons le résultat dans une ArrayCollection
 - L'API est disponible à cette adresse :
<http://developer.yahoo.com/search/local/V3/localSearch.html>
 - En se connectant à la main, on voit qu'il faut préciser 3 arguments pour obtenir un résultat : deux sont obligatoires :
 - **appid** : applicationID, identifiant de manière unique votre application
 - On peut en créer un ou utiliser YahooDemo
 - **query** : le texte recherché



HTTPService

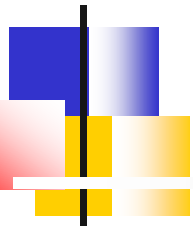
■ Pour accéder maintenant aux données

- Nous recherchons les restaurants de San Francisco dont l'un des codes postaux est **94108**
- Nous allons donc construire notre requête à partir de l'URL de base du service

<http://local.yahooapis.com/LocalSearchService/V3/localSearch>

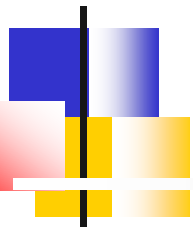
- puis, on ajoute les 3 paramètres suivants :
 - `appid=YahooDemo`
 - `query=restaurant`
 - `zip= 94108`
- ce qui donne en mode **GET**

<http://local.yahooapis.com/LocalSearchService/V3/localSearch?appid=YahooDemo&query=restaurant&zip=94108>



HTTPService

- A présent, lancez le résultat dans un navigateur et observez le résultat final
- La requête retourne un flux XML contenant une mine d'informations comme :
 - Le nombre total de résultats
 - Le nombre de résultats retournés
 - Un lien Yahoo! Maps
- Et pour chaque restaurant
 - Les coordonnées (adresse, téléphone...)
 - Une note...



HTTPService

■ Comment le faire à travers Flex ?

- Voici les étapes de création :

1. Créez une nouvelle application
2. Ajoutez un bouton pour appeler la requête et 5 champs pour afficher les informations de base :

```
<mx:Label id="Nom"/>
```

```
<mx:Label id="Adresse"/>
```

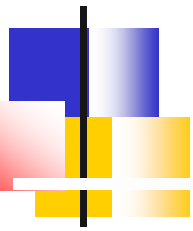
```
<mx:Label id="Telephone"/>
```

```
<mx:Label id="Note"/>
```

```
<mx:Label id="URL"/>
```

```
<mx:Button label="Lancer la recherche" click =  
"appel()"/>
```

```
</mx:Application>
```



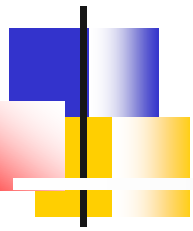
HTTPService

3. Ajoutez maintenant le HTTPService que nous nommerons **restaurant** :
 - Il faut ajouter la balise `<mx:request>` pour donner les paramètres de la requête

```
<mx:HTTPService id="restaurant"  
    url="http://local.yahooapis.com/LocalSearchService/V3/localSearch"
```

```
>
```

```
<mx:request>  
    <appid>YahooDemo</appid>  
    <query>restaurant</query>  
    <zip>94108</zip>  
</mx:request>
```



HTTPService

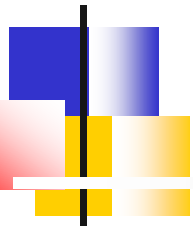
4. Maintenant que la requête est définie, il faut l'appeler et stocker le résultat...
 - Pour appeler une requête http, il faut invoquer la méthode `send()` d'un objet de type `HTTPService`, ici: `restaurant`

```
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="vertical"
  creationComplete="restaurant.send()"
>
```

- Les résultats sont récupérés dans la propriété `lastResult` de l'objet `HTTPService`

- Le code : localSearch.mxml
 - Result[0] indique que l'on ne récupère que le premier restaurant

```
<mx:Script>
<![CDATA[
    public function appel() : void{
        Nom.text= restaurant.lastResult.ResultSet.Result[0].Title;
        Adresse.text= restaurant.lastResult.ResultSet.Result[0].Address;
        Telephone.text= restaurant.lastResult.ResultSet.Result[0].Phone;
        Note.text=
        restaurant.lastResult.ResultSet.Result[0].Rating.AverageRating;
        URL.text= restaurant.lastResult.ResultSet.Result[0].BusinessUrl;
    }
]}>
</mx:Script>
<mx:HTTPService...
```

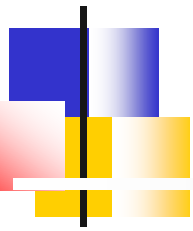



HTTPService

5. On obtient le résultat correspondant au premier restaurant

Lancer la recherche

Gary Danko
800 N Point St
(415) 749-2060
4.5
<http://www.garydanko.com/>



HTTPService

- Recherche de plusieurs restaurants avec
 - saisie de la requête dans un formulaire
 - rangement des restaurants dans un autre formulaire
- Pour rentrer les paramètres de la requête par saisie dans un formulaire,
 - on utilise le binding :
 - qui liera les champs du formulaire aux paramètres contenus dans la balise `<mx:Request>` du HTTPService
- Comment récupérer les résultats de la requête et les ranger dans un formulaire ?
 - on utilise le binding
 - qui lie `ResultSet.Result` du XML renvoyé en réponse avec la `DataGrid`

Affichage : binding

```
<mx:DataGrid
  dataProvider="{restaurant.lastResult.ResultSet.Result}" width="900"
  height="300">
  <mx:columns>
    <mx:DataGridColumn
      dataField="Address"/>
    <mx:DataGridColumn
      dataField="Title"/>
    <mx:DataGridColumn
      dataField="BusinessClickUrl"
      headerText="URL"/>
    <mx:DataGridColumn
      dataField="City"/>
    <mx:DataGridColumn
      dataField="Phone"/>
  </mx:columns>
</mx:DataGrid>

<mx:Button label="Lancer la recherche"
  click="restaurant.send()"/>
</mx:Application>
```

Lancement

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

  <mx:HTTPService id="restaurant"
    url="http://local.yahooapis.com/LocalSearchService
      /V3/localSearch">

    <mx:request>
      <appid>YahooDemo</appid>
      <query>{formQuery.text}</query>
      <zip>{formZip.text}</zip>
    </mx:request>
  </mx:HTTPService>

  <mx:Form label="paramètres de la requête">
    <mx:FormItem label="Query">
      <mx:TextInput id="formQuery"/>
    </mx:FormItem>
    <mx:FormItem label="Code postal">
      <mx:TextInput id="formZip"/>
    </mx:FormItem>
  </mx:Form>
```

Requête : binding

Saisie

restaurant : sera le mot rentré lors de la saisie

HTTPService

■ Exemple : lectureURLBinding.mxml

Query

Code postal

Address	Title	URL	City	Phone
800 N Point St	Gary Danko	http://www.garydanko.com/	San Francisco	(415) 749-2060
2299 Powell St	Caesar's Italian Restaurant	http://caesars.citysearch.com/	San Francisco	(415) 989-6000
1031 Irving St	San Tung Chinese Restaurant		San Francisco	(415) 242-0828
309 Clement St	Burma Super Star	http://www.burmasuperstar.com	San Francisco	(415) 387-2147
160 Ellis St	New Delhi Restaurant		San Francisco	(415) 397-8470
199 Valencia St	Zeitgeist	http://www.sonic.net/~wwpints/	San Francisco	(415) 255-7505
47 Pier, #1	Scoma's	http://www.scomas.com/	San Francisco	(415) 771-4383
217 Columbus Ave	Brandy Ho's Hunan Food	http://www.brandyhos.com/	San Francisco	(415) 788-7527
8 6th St	Tu Lan		San Francisco	(415) 626-0927
225 11th St	Don Ramon's Mexican Restaur	http://donramons.ypguides.net	San Francisco	(415) 864-2700







Lancer la recherche

N'oubliez pas de remplir ce formulaire

■ Un autre exemple : resultFormat_e4x.mxml

- permettant d'accéder à une galerie d'images sur le site :
 - http://api.flickr.com/services/feeds/photos_public.gne
- Le résultat est récupéré en e4x
 - On récupère son nom, son code et un lien vers le site où elle se trouve

Public photos

		
6	P1020645	Thank you, Mr. Awesome!
lizhaemma	MnGyver	infinite.p
		
P1210200	IMG_5194	Liz & Jim's Wedding 448
alcanina	ourSunPhotos	TruthCraze

- Le HTTPService

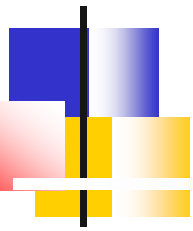
```
<mx:HTTPService id="photoService"
    url="http://api.flickr.com/services/feeds/photos_public.gne"
    resultFormat="e4x"
    //événement quand le chargement est terminé
    result="photoResultHandler(event);"
    // événement quand une erreur se produit
    fault="photoFaultHandler(event);"
/>
```

- Le résultat est donné par la fonction :

```
private function photoResultHandler(event:ResultEvent):void{
    photoFeed = event.result as XML;
}
```

- L'interface

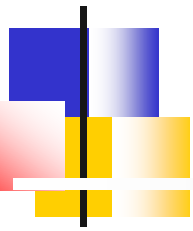
```
<mx:Panel title="Public photos" width="100%" height="100%">
  <mx:Tile width="100%" height="100%">
    <mx:Repeater id="photos" dataProvider="{photoFeed.entry}">
      <mx:VBox ...>
        <mx:Image id="myImage"
          // récupération de la photo
          source="{parseImageUrl(photos.currentItem.content)}"
          // lui appliquer un effet
          completeEffect="{fadeIn}"/>
          //récupération du titre
        <mx:Text text="{photos.currentItem.title}"/>
          //bouton pour avoir plus d'info sur une photo
        <mx:LinkButton
          label="{photos.currentItem.author.name}"
          click="openAuthorPage(event);"
        />
      </mx:VBox>
    </mx:Repeater>
  </mx:Tile>
  <mx:ControlBar horizontalAlign="center">
    <mx:Button label="Update" click="photoService.send();"/>
  </mx:ControlBar>
</mx:Panel>
```



HTTPService

■ Gérer les erreurs

- Une requête peut échouer pour diverses raisons :
 - serveur injoignable, attente dépassée...
- Flex propose d'intercepter les erreurs grâce à un gestionnaire d'événements
- A chaque erreur déclenchée lors d'un appel de `send()`, un événement de type `FaultEvent` est envoyé
- Ce dernier contient :
 - Un objet type `Fault` (propriété `fault`) de la classe `Error` et contenant le détail de l'incident (`FaultCode`, `FaultDetail`, `FaultString` et `RootCause`)
 - Un message (contenant l'ensemble des informations précédentes)
- C'est l'événement `fault` d'un `HTTPService` qui permet d'intercepter ce type d'erreur



HTTPService

■ Exemple : localSearchError.mxml

- Il s'agit d'afficher un pop-up en cas d'erreur lors de l'envoi de la requête

■ Étapes

1. Créer une méthode ActionScript prenant en paramètre un objet de type FaultEvent

```
import mx.controls.Alert;
```

```
import mx.rpc.events.FaultEvent;
```

```
import mx.rpc.events.ResultEvent;
```

```
//on affiche tous les détails de l'erreur produite
```

```
public function faultHandler(evt:FaultEvent):void{
```

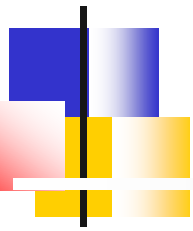
```
    Alert.show("une erreur s'est produite\n :" + "\nDetails\n" +
```

```
    "faultCode : " + evt.fault.faultCode + "\n" +
```

```
    "faultDetail : " + evt.fault.faultDetail + "\n" +
```

```
    "faultString : " + evt.fault.faultString + "\n");
```

```
}
```

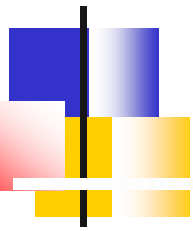


HTTPService

2. Affecter cette méthode à l'événement **fault** de votre HTTPService

```
<mx:HTTPService id="restaurant"
  url="http://local.yahooapis.com/LocalSearchService/
  V3/localSearch" result="processResult(event)"
  fault="faultHandler(event)">
<mx:request>
  <appid>YahooDemo</appid>
  <query>{formQuery.text}</query>
  <zip>{formZip.text}</zip>
</mx:request>
</mx:HTTPService>
```



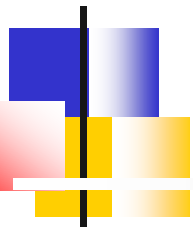


HTTPService

- Résultat
 - Le gestionnaire d'événement sort une erreur car il s'aperçoit que le champ **query** n'a pas été rempli
 - Dans la version précédente, il attendait que le champ soit rempli à la main

```
une erreur s'est produite
:
Details
faultCode : Server.Error.Request
faultDetail : Error: [IOErrorEvent type="ioError"
bubbles=false cancelable=false eventPhase=2
text="Error #2032"]. URL:
http://local.yahooapis.com/LocalSearchService/V
3/localSearch
faultString : HTTP request error
```

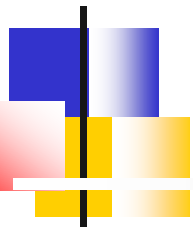
OK



Services Web

■ Fonctionnement du service Web

- Similaire à celui des HTTPServices
- Cependant, pour une adresse donnée, il peut exister plusieurs méthodes différentes
- Exemple :
 - Un serveur proposant des informations sur la finance
 - Cours d'un indice ou
 - Cours d'une matière première
 - Taux de change entre monnaies...
 - Il pourrait donc exister une méthode par type d'information recherchée
- Un service Web se définit donc en Flex par :
 - L'adresse du document **wSDL** décrivant le service à préciser dans :
 - propriété **wSDL**
 - Une ou plusieurs méthodes à appeler à préciser dans la balise :
 - balise **<mx:operation>**



Services Web

■ Exemple : webServiceMeteo.mxml

- Cet exemple permet de récupérer les données météo d'une semaine, sur un site de météo pour les states aux U.S.
- Nous utilisons le service situé à l'adresse :
 - <http://www.webservices.net/WeatherForecast.asmx>
- La méthode prend en paramètre un seul argument, le **zipcode** de l'état

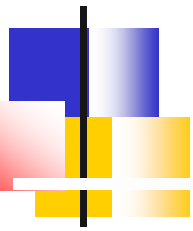
WebService Example

```
<GetWeatherByZipCodeResponse
xmlns="http://www.webservices.net"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <GetWeatherByZipCodeResult>
    <Latitude>33.57975</Latitude>
    <Longitude>85.08121</Longitude>
    <AllocationFactor>0.006228</AllocationFactor>
    <FipsCode>13</FipsCode>
    <PlaceName>CARROLLTON</PlaceName>
    <StateCode>GA</StateCode>
    <Details>
      <WeatherData>
        <Day>Tuesday, July 28, 2009</Day>
      </WeatherData>
    </GetWeatherByZipCodeResult>
  </GetWeatherByZipCodeResponse>
```

Enter a zip code.

CARROLLTON, GA

Day	High	Low
Tuesday, July 28, 2009	86	69
Wednesday, July 29, 2009	82	69
Thursday, July 30, 2009	87	70
Friday, July 31, 2009	86	69
Saturday, August 01, 2009	86	68
Sunday, August 02, 2009	87	69



Web service

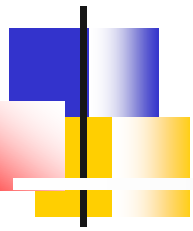
■ Les étapes

1. Créez une nouvelle application WebServiceMeteo et déclarez y un composant WebService comme ceci :

```
<mx:WebService id="WS"
  wsdl="http://www.webservices.net/WeatherForecast.asmx?WSDL"
</mx:WebService>
```

- Vous pouvez, comme un HTTPService, définir un certain nombre de propriétés ou de méthodes à appeler sur des événements spécifiques

```
<mx:WebService id="WS" //id : nom de l'instance de l'objet WebService
  //emplacement du service web à utiliser
  wsdl="http://www.webservices.net/WeatherForecast.asmx?WSDL"
  useProxy="false"
  //gestionnaire d'événement appelé lorsque les données sont
  //retournées par le service web
  fault="Alert.show(event.fault.faultString), 'Error'"
  //gestionnaire d'événement à appeler en cas d'erreur retournée par
  //le service web
  result="onResult(event)" >
</mx:WebService>
```



Web service

- Quand on appelle le service :
<http://www.webservices.net/WeatherForecast.asmx>
- Il nous apprend qu'on peut donner l'un des deux paramètres :
 - **valid zip code** or **Place name**
- En effet, voilà ce que donne la page :
 - The following operations are supported. For a formal definition, please review the Service Description
 - **GetWeatherByPlaceName**
 - Get one week weather forecast for a place name(USA)
 - **GetWeatherByZipCode**
 - Get one week weather forecast for a valid Zip Code(USA)



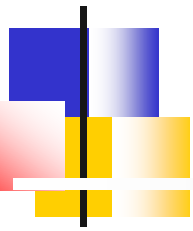
Web service

2. Choisir l'opération :

- la seule opération permise par le service est `GetWeatherByZipCode`

```
<mx:WebService id="WS"  
  wsdl="http://www.webservices.net/WeatherForecast.asmx?WSDL"  
  useProxy="false"  
  fault="Alert.show(event.fault.faultString), 'Error'"  
  result="onResult(event)" >
```

```
<mx:operation name="GetWeatherByZipCode" resultFormat="e4x" >  
  <mx:request>  
    ...  
  </mx:request>  
</mx:operation>  
</mx:WebService>
```

Web service

3. Passer des paramètres à la méthode

- De la même manière que pour un HTTPService, il est possible de passer des arguments au service web de deux manières :
 - Par la liaison de données dans <mx:request>
 - Directement dans l'appel de la méthode en AS
- Ici, on choisit la première :

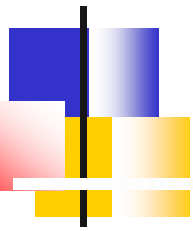
```
<mx:operation name="GetWeatherByZipCode"
  resultFormat="e4x" >
  <mx:request>
    <ZipCode>{zipcode.text}</ZipCode>
  </mx:request>
</mx:operation>
</mx:WebService>
```

4. Compléter l'application pour pouvoir appeler le service web correctement

```
<mx:WebService id="WS" wsdl="http://www.webservices.net/WeatherForecast.asmx?WSDL"
    useProxy="false" fault="Alert.show(event.fault.faultString), 'Error'" result="onResult(event)"
    >
    <mx:operation name="GetWeatherByZipCode" resultFormat="e4x" >
        <mx:request><ZipCode>{zipcode.text}</ZipCode></mx:request>
    </mx:operation>
</mx:WebService>
```

```
<mx:Panel title="WebService Example" height="75%" paddingTop="10"
    paddingBottom="10" paddingLeft="10" paddingRight="10" id="myPanel">
<mx:TextArea x="200" width="400" height="250" id="outputInfo" />
<mx:HBox>
    <mx:Label width="100%" color="blue" text="Enter a zip code." />
    <mx:TextInput id="zipcode" text="30117" />
    <mx:Button label="Get weather" click="getWeather()" />
</mx:HBox>
```

```
<mx:Text id="txtPlace" htmlText="{_sPlace}" />
<mx:DataGrid dataProvider="{_xlDayData}" height="180">
    <mx:columns>
        <mx:DataGridColumn labelFunction="IfDayData" headerText="Day" width="200" />
        <mx:DataGridColumn labelFunction="IfDayData" headerText="High" width="100" />
        <mx:DataGridColumn labelFunction="IfDayData" headerText="Low" width="100" />
    </mx:columns>
</mx:DataGrid></mx:Panel>
```



Web Services

■ Commentaires

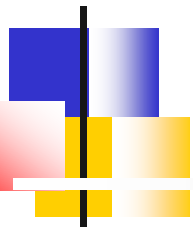
- Nous avons seulement lié les paramètres de la requête avec les champs `TextInput` et le bouton qui appelle la méthode `getWeather()`
- Les résultats de cette requête sont directement liés aux champs

- Text et dataGrid

```
<mx:Text id="txtPlace" htmlText="{_sPlace}"/>  
<mx:DataGrid dataProvider="{_xlDayData}"  
  height="180">
```

- avec

```
_sPlace = xmlResultNode.PlaceName.text() + ", " +  
  xmlResultNode.StateCode.text();  
_xlDayData = xmlDetailsNode.WeatherData;
```



Web Services

- Différence d'utilisation entre Webservice et HTTPService

Méthode d'accès aux données

	Demande de données	Résultat
HTTPService	serviceId.send()	serviceId.lastResult
Webservice	serviceId. methodId .send()	serviceId. methodId .lastResult

Adresse des ressources

	Propriété	Description
HTTPService	url=	adresse web de la ressource demandée
Webservice	WsdL=	adresse du document wSDL de description du service

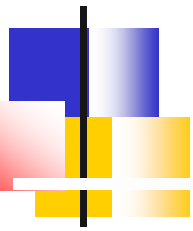


Web Services

- Différence d'utilisation entre WebService et HTTPService (suite)

Paramètres et méthodes

	Demande de données	Résultat
HTTPService	Méthodes : 1 paramètre (<mx:request>)	Le composant ne fournit un accès qu'à une unique ressource externe
WebService	Méthodes : autant que de méthodes décrites dans le fichier wsdl Paramètres : n (<mx:request>)	Le composant permet de définir plusieurs méthodes au sein du même service web par le biais de la balise <mx:operation>



Connexion avec une BD

■ Fonctionnement

- On passe par PHP qui se connecte à une base de données MySql
- PHP fournit à Flex, via un HTTPService, du code XML
- Les balises XML correspondant aux noms sont placées dans les différentes colonnes de la table

■ Étapes

1. Créer la base MySQL
2. Copier le code PHP ci-après et ajuster les paramètres de connexion et d'interrogation de la base
3. Ouvrir le fichier PHP dans un navigateur pour vérifier l'affichage correct de XML (voir le source)
4. Copier le code Flex (**FlexBD.mxml**) et ajuster les paramètres de l'URL pour se connecter au script PHP
5. Ajuster le nom de colonne dans la datagrid pour correspondre au nom de colonne dans le XML
6. Exécuter l'application Flex

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="absolute" initialize="init()">
<mx:Script>
<![CDATA[
  import mx.rpc.events.ResultEvent;
  import mx.collections.ArrayCollection;
  [Bindable] ← Indique que dataList sera
  private var dataList:ArrayCollection;           liée à une structure interne
  private function onResult(event:ResultEvent):void{ ← Récupération en mode
    dataList = event.result.data.row; ← asynchrone dans une
  }                                               dataList
  private function init():void{
    database.send(); ← Récupération des données
  }
  ]]>
</mx:Script>

<mx:HTTPService
  id="database"
  url="http://localhost/link.php" ← Connexion avec PHP
  result="onResult(event)"
  showBusyCursor="true"
/>

```

La datalist (déclarée Bindable) est liée avec la DataGrid



```
<mx:DataGrid id="dataView" dataProvider="{dataList}">
  <mx:columns>
    <mx:DataGridColumn headerText="Nom"
      dataField="nom"/>
    <mx:DataGridColumn headerText="Prénom"
      dataField="prenom"/>
    <mx:DataGridColumn headerText="Sexe"
      dataField="sexe"/>
  </mx:columns>
</mx:DataGrid>
</mx:Application>
```


Le fichier `link.php` à mettre dans le répertoire `www` de EasyPhp

```
<?php
$connection = mysql_connect('localhost', 'root', 'mysql') or die ('cannot
    reach database');
```

Mes paramètres

```
//Connexion à la base "test"
```

```
$db = mysql_select_db("test") or die ("this is not a valid database");
```

```
//Lecture de la base test_mxml
```

```
$result = mysql_query("SELECT * FROM test_mxml");
```

```
//Calcul du nombre de lignes
```

```
$num_row = mysql_num_rows($result);
```



//Préparation de la sortie XML

```
echo '<?xml version="1.0" encoding="iso-8859-1"?>';  
echo "<data>";  
echo '<num>' . $num_row . '</num>';  
if (!$result) {  
    die('Query failed: ' . mysql_error());  
}
```

// lecture des meta-données et des noms des colonnes

```
$i = 0;  
while ($i < mysql_num_fields($result)) {  
    $meta = mysql_fetch_field($result, $i);  
    $ColumnNames[] = $meta->name; //place col name dans array  
    $i++;  
}  
$specialchar = array("&", ">", "<"); //Caractères spéciaux  
$specialcharReplace = array("&"; ">"; "<"); //remplacement
```

```
/* Conversion des données de la table et des noms de colonnes en XML*/
```

```
$w = 0;
```

```
while ($line = mysql_fetch_array($result, MYSQL_ASSOC)) {
```

```
    echo "<row>";
```

```
    foreach ($line as $col_value){
```

```
        echo '<'.$ColumnNames[$w].>';
```

```
        $col_value_strip = str_replace($specialchars, $specialcharsReplace,  
        $col_value);
```

```
        echo $col_value_strip;
```

```
        echo '</'.$ColumnNames[$w].>';
```

```
        if($w == ($i - 1)) {$w = 0;}
```

```
        else { $w++; }
```

```
    }
```

```
    echo "</row>";
```

```
}
```

```
if($num_row == "1"){ echo '<row></row>';}
```

```
echo "</data>";
```

```
mysql_free_result($result);
```

```
?>
```

N'oubliez pas de lancer le serveur

est (1)

test_mxml

```
FROM `test_mxml`  
LIMIT 0 , 30
```

Profilage [Modifier] [Expliquer SQL]

Afficher : 30 enregistrement(s) à partir de l'enregistrement n° 0
en mode horizontal et répéter les en-têtes à chaque groupe

Trier sur l'index: aucune

+ Options

	id	nom	prenom	sexe
<input type="checkbox"/>	1	RARD	David	1
<input type="checkbox"/>	2	John	Doe	1
<input type="checkbox"/>	3	Tom	Pousse	1
<input type="checkbox"/>	4	V	Vendetta	2
<input type="checkbox"/>	5	test	testort	-1

Tout cocher / Tout décocher Pour la sélection :

Afficher : 30 enregistrement(s) à partir de l'enregistrement n° 0
en mode horizontal et répéter les en-têtes à chaque groupe

Nom	Prénom	Sexe
RARD	David	1
John	Doe	1
Tom	Pousse	1
V	Vendetta	2
test	testort	-1

- Création d'un projet Flex avec connexion directe avec PHP : suivre cette démarche

Create a Flex project.
Choose a name and location for your project, and configure the server technology your project will be using.

Project name: **BDProject** **nom du projet**

Project location
 Use default location
Folder: **C:\Program Files\EasyPHP 3.0\www** **Chemin du WWW de PHP**

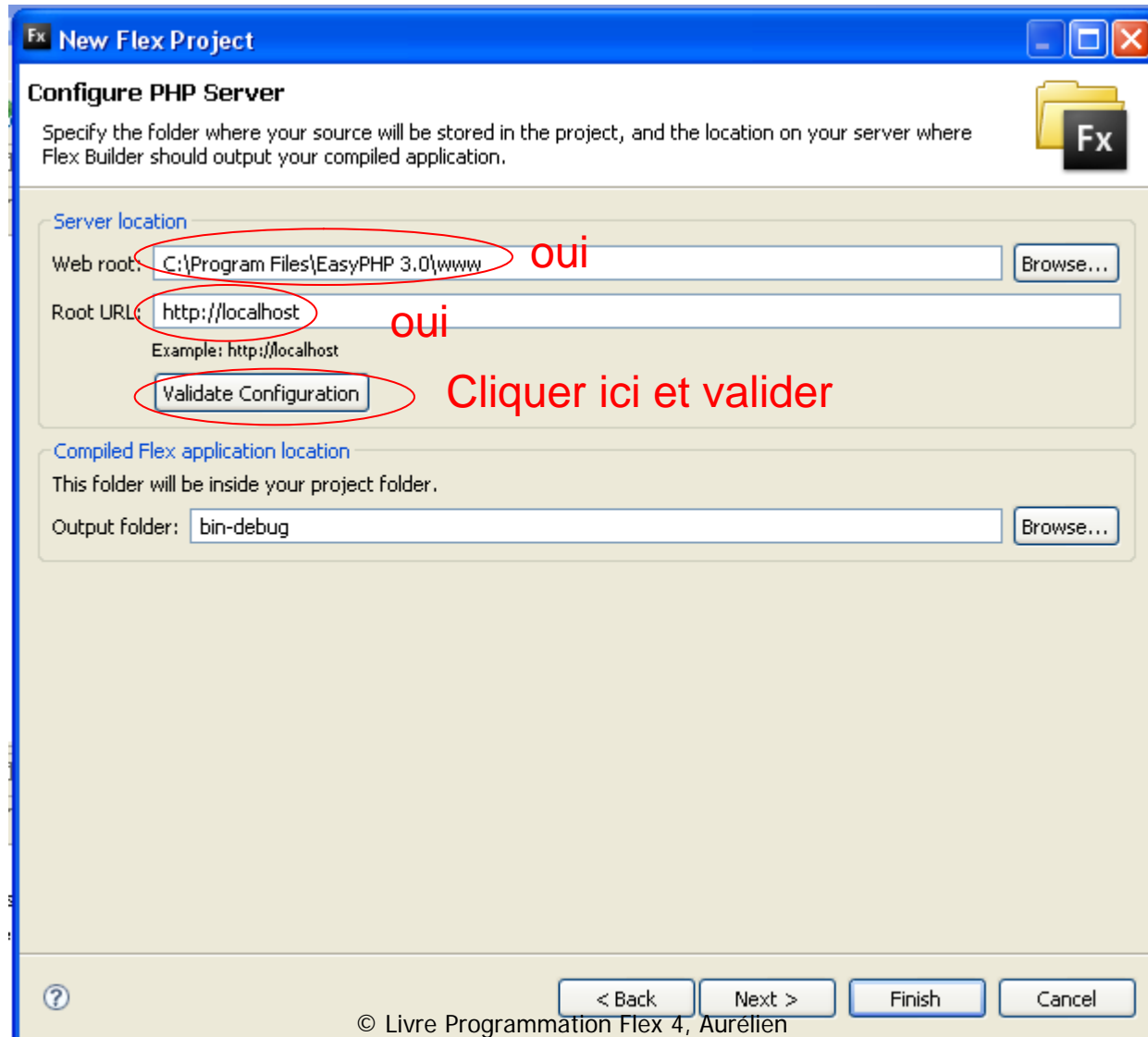
Application type
 Web application (runs in Flash Player)
 Desktop application (runs in Adobe AIR)

Server technology
Application server type: **PHP** **nom du serveur**
 Use remote object access service
 LiveCycle Data Services
 ColdFusion Flash Remoting

< Back **Next >** Finish Cancel

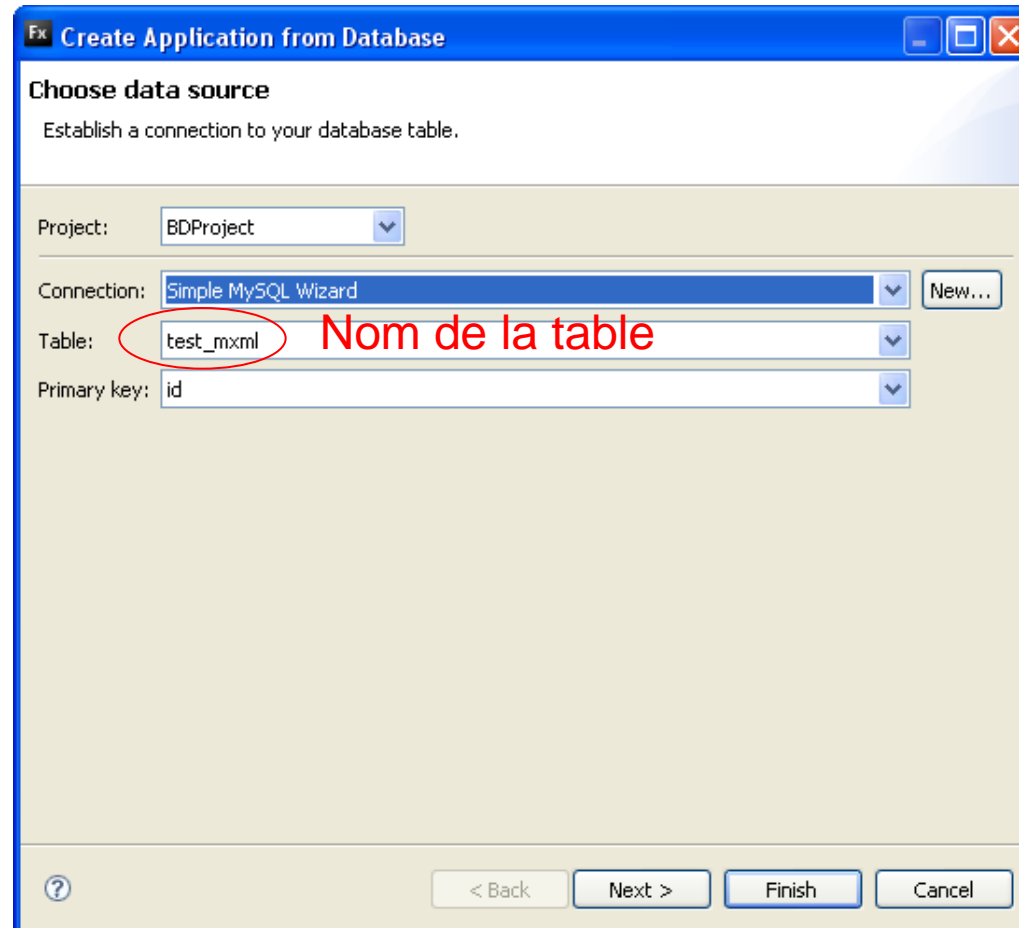
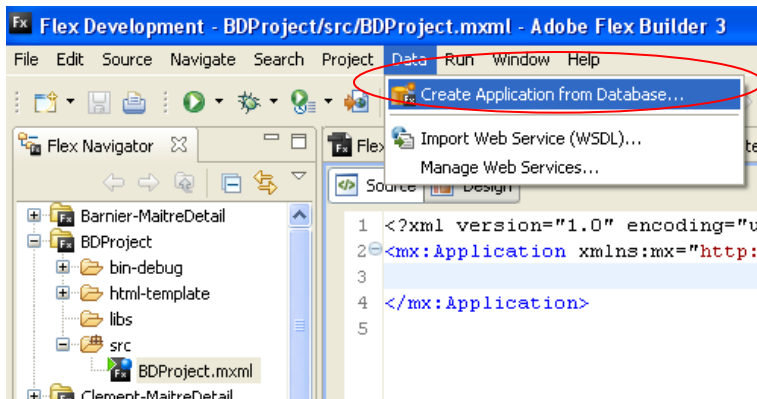
■ Le projet Flex : BDProject

- Accepter la configuration proposée par le système

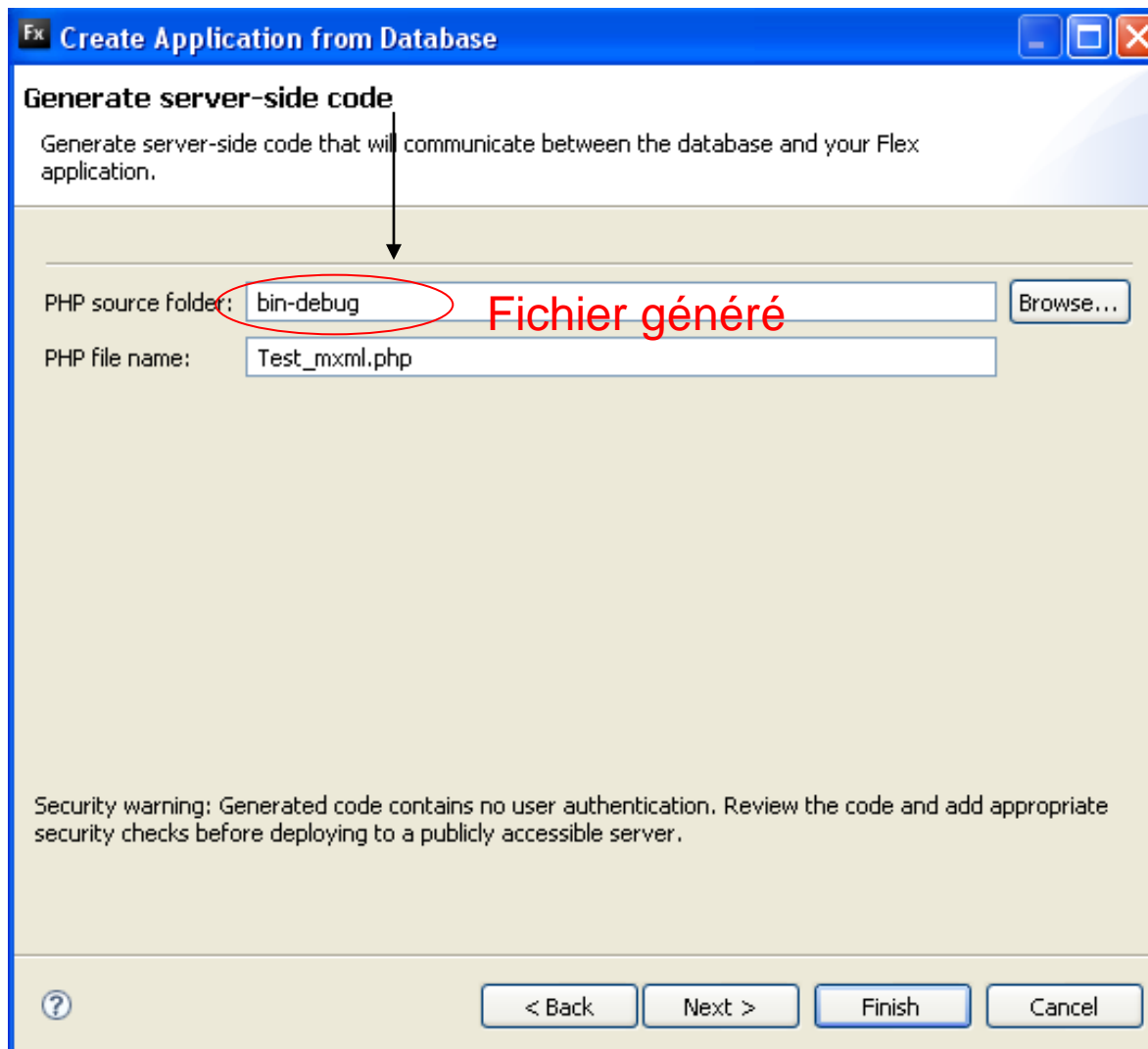


■ Création d'un projet Flex :

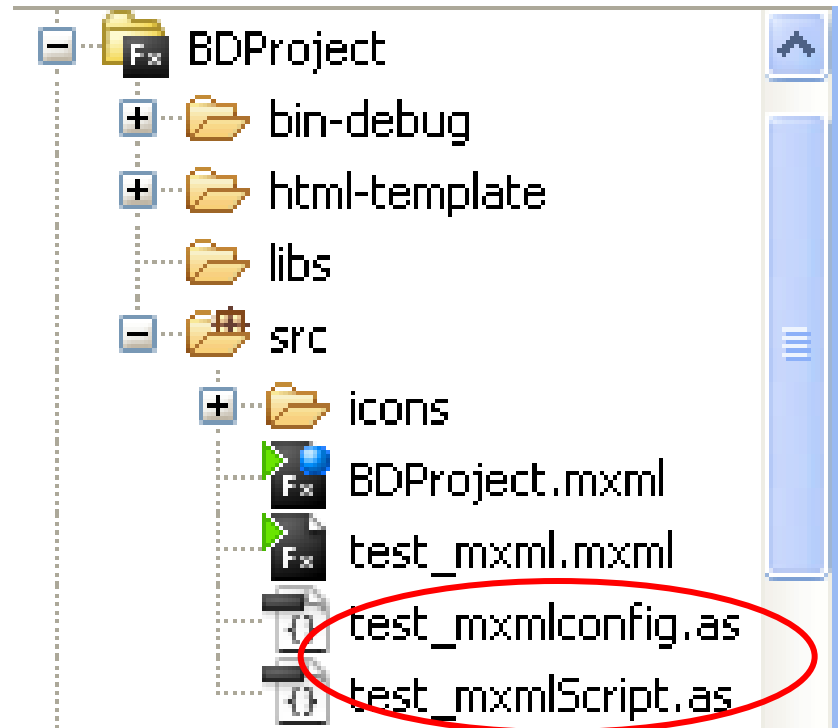
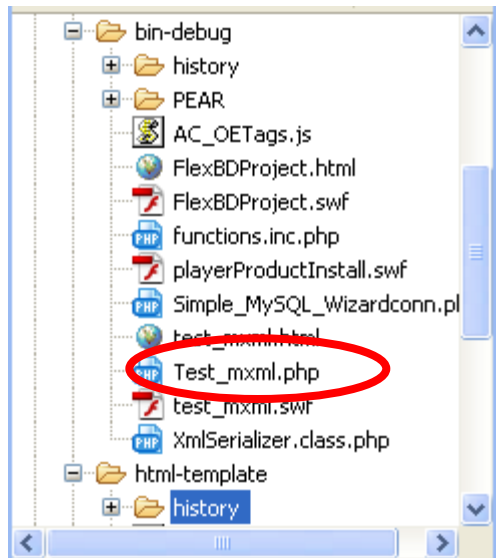
- Se mettre sur le projet
- Créer une connexion avec la base



- Il génère automatiquement un fichier de lecture de la base
 - Test_mxml.php

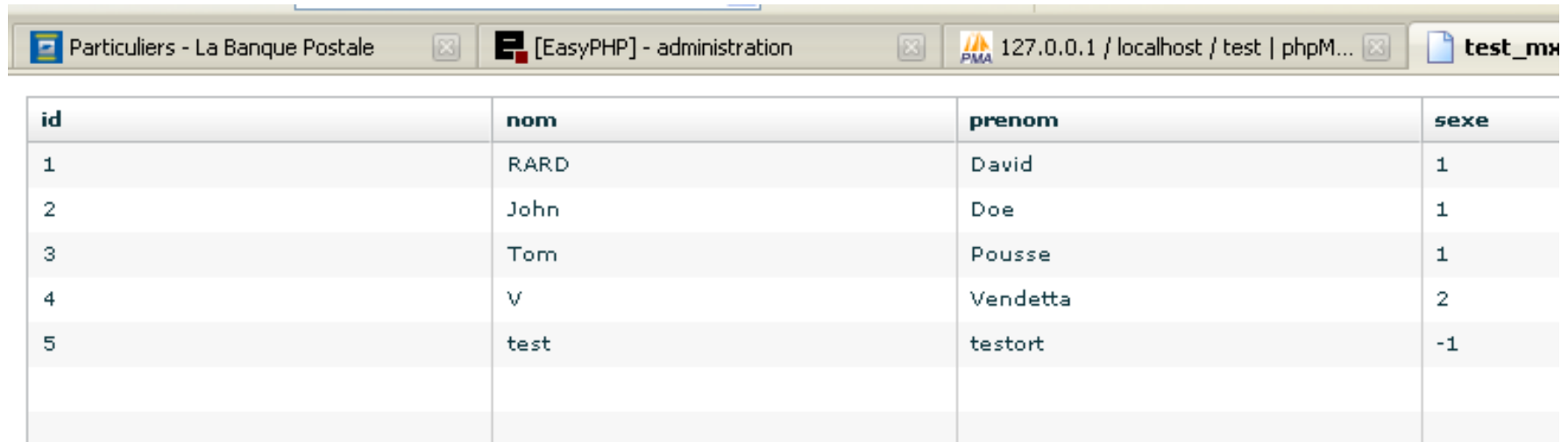


- Il génère également automatiquement 2 autres fichiers



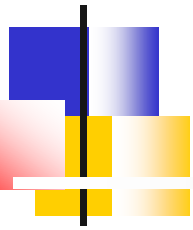
■ Si on exécute Test_mxml.mxml

- On voit que Flex récupère la base sous la forme d'une DataGrid



The screenshot shows a browser window with three tabs: 'Particuliers - La Banque Postale', '[EasyPHP] - administration', and '127.0.0.1 / localhost / test | phpM...'. The main content area displays a DataGrid table with the following data:

id	nom	prenom	sexe
1	RARD	David	1
2	John	Doe	1
3	Tom	Pousse	1
4	V	Vendetta	2
5	test	testort	-1



Exercice

■ Énoncé

- Créer un HTTPService avec connexion avec PHP de traduction traducteurFA.mxml, en utilisant une base mySql