3 Application client/serveur

L'exemple qui suit met en œuvre une application Flex qui accède à un fichier XML distant. Le contenu du fichier étant utilisé dans le but de peupler une grille de données.

Jusqu'à maintenant, nous avons exécuté nos applications Flex en local. L'objectif est de distribuer ces applications et pour ce faire, nous allons utiliser un serveur web.

3.1 Pré requis

Il est nécessaire d'installer un serveur **Apache** ou tout autre serveur web équivalent.

Dans la suite du document, ***{HTDOCS}*** désigne le chemin du répertoire racine des fichiers du serveur Web (Par exemple *C:/easyphp/www*). Quant à ***{SERVER URL}***, il désigne l'url utilisée pour accéder aux fichiers du serveur (Par exemple *http://localhost*).

3.2 Création de la ressource distante

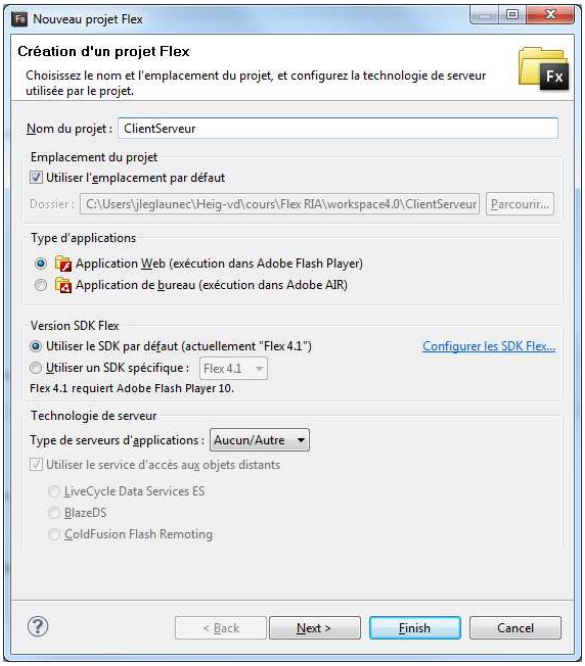
Pour commencer, le fichier **users.xml** doit être déposé dans le dossier ***{HTDOCS} /clientserveur*** du serveur et accessible par une URL semblable à ***{SERVER URL} /clientserveur/users.xml***.

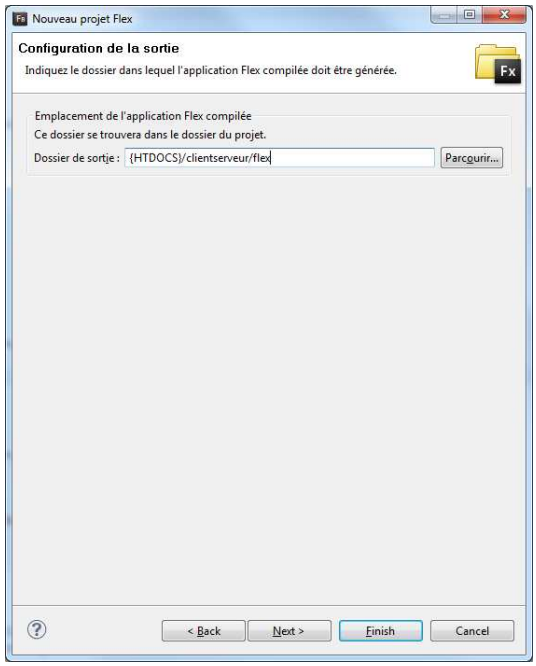
users.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<users>
    <user id="1">
        <name>User 1</name>
        <birthDate>1980-01-01</birthDate>
    </user>
    <user id="2">
        <name>User 2</name>
        <birthDate>1985-04-23</birthDate>
    </user>
    <user id="3">
        <name>User 3</name>
        <birthDate>1986-07-09</birthDate>
    </user>
    <user id="4">
        <name>User 4</name>
        <birthDate>1986-03-19</birthDate>
    </user>
</users>
```

3.3 Création du projet

Revenons à **Flex Builder** afin de créer un nouveau projet.





Attention de bien modifier le champ **Output folder** afin qu'il pointe dans un répertoire du serveur Web. ***{HTDOCS} /clientserveur/flex***

3.4 Chargement du fichier XML distant

Pour charger un fichier XML distant, nous utilisons le composant `HTTPService` avec les paramètres suivants :

Service distant récupérant un fichier XML

```
<s:HTTPService
    id="hseService"
    url="http://localhost/clientserveur/users.xml"
    result="onResult(event)"
    fault="onFault(event)"
    showBusyCursor="true">
```

On transmet l'url de la ressource, le format attendu en retour ainsi que les méthodes de callback associées.

Définition des méthodes de callback en ActionScript

```
[Bindable]
private var remoteXmlFile : XML;

public function onResult(event : ResultEvent) : void
{
    remoteXmlFile = new XML(event.message.body);
}
```

```
public function onFault(event : FaultEvent) : void
{
    Alert.show("error :"+event.message);
}
```

La variable `remoteXmlFile` est précédée de la métadonnée `[Bindable]`. Cela implique que le compilateur Flex va générer automatiquement un événement `propertyChange`. Ceci est indispensable étant donné que la construction de l'interface et le chargement du fichier sont asynchrones. Il est nécessaire qu'un moment donné, l'application informe les composants visuels que la variable a été modifiée et, en d'autres termes, que le fichier a été chargé.

Il suffit maintenant d'appeler la méthode `hseService.send()` à l'initialisation de l'application afin d'envoyer la requête qui nous permettra d'obtenir le fichier distant.

Appel du fichier distant

```
<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"
               initialize="{hseService.send()}">
```

3.5 Affichage des utilisateurs dans une liste

Maintenant que le fichier XML des utilisateurs a été chargé dans l'application Flex, nous allons afficher son contenu dans un composant **DataGrid**.

Pour ce faire, ajouter le composant suivant

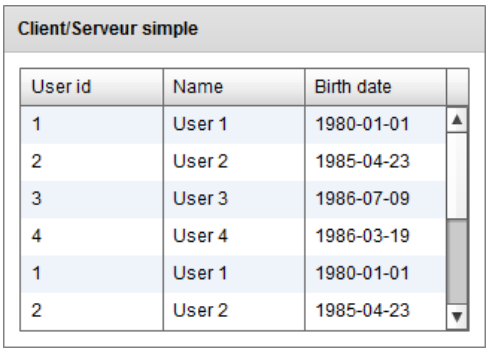
```
<mx:DataGrid
    id="dgdUsers"
    dataProvider="{remoteXmlFile.user}"
    width="100%">
    <mx:columns>
        <mx:DataGridColumn
            dataField="@id"
            headerText="User id"/>
        <mx:DataGridColumn
            dataField="name"
            headerText="Name"/>
        <mx:DataGridColumn
            dataField="birthDate"
            headerText="Birth date"/>
    </mx:columns>
</mx:DataGrid>
```

On remarque que le `dataProvider` fait référence aux utilisateurs présents dans le fichier chargé à l'étape précédente. Les `columns` associent un élément du XML à une colonne de la grille.

3.6 Résultat

Avant d'exécuter l'application, il est nécessaire de se rendre dans l'interface *« Run configurations »* afin de modifier le champ *« URL ou chemin à lancer ».* Dans notre cas, introduire *« {SERVER URL}/clientserveur/flex/ClientServeur.html »*

L'application **ClientServeur** produit l'affichage suivant :



Le code source complet est disponible ci-dessous :

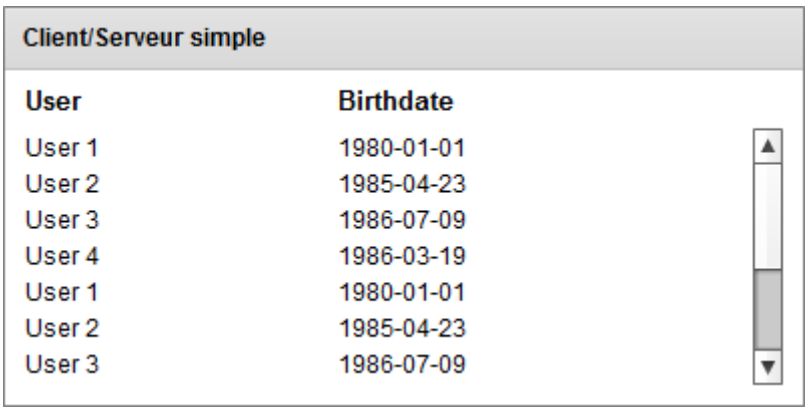
ClientServeur.mxml

```
<?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"
               initialize="{hseService.send()}">
    <fx:Declarations>
        <!-- Service distant récupérant un fichier XML -->
        <s:HTTPService
            id="hseService"
            url="http://localhost/clientserveur/users.xml"
            result="onResult(event)"
            fault="onFault(event)"
            showBusyCursor="true"/>
    </fx:Declarations>
    <fx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.rpc.events.FaultEvent;
            import mx.rpc.events.ResultEvent;

            [Bindable]
            private var remoteXmlFile : XML;

            private function onResult(event : ResultEvent) : void
            {
                remoteXmlFile = new XML(event.message.body);
            }

            private function onFault(event : FaultEvent) : void{
                Alert.show("error : "+event.message);
            }
        ]]>
    </fx:Script>
    <!-- Panel principal -->
    <s:Panel
```

```
        id="pnlClientServerSimple"
        title="Client/Serveur simple"
        verticalCenter="1"
        horizontalCenter="1">
        <s:layout>
            <s:VerticalLayout horizontalAlign="center" paddingBottom="10"
paddingLeft="10" paddingRight="10" paddingTop="10"/>
        </s:layout>
        <mx:DataGrid
            id="dgdUsers"
            dataProvider="{remoteXmlFile.user}"
            width="100%">
            <mx:columns>
                <mx:DataGridColumn
                dataField="@id"
                headerText="User id"/>
                <mx:DataGridColumn
                dataField="name"
                headerText="Name"/>
                <mx:DataGridColumn
                dataField="birthDate"
                headerText="Birth date"/>
            </mx:columns>
        </mx:DataGrid>
    </s:Panel>
</s:Application>
```

3.7 Exercice

Créer une application similaire toujours basée sur le fichier XML ***users.xml***. Cette fois-ci, on n'utilisera pas l'objet DataGrid pour afficher le contenu du fichier XML mais uniquement des labels créés « *on-the-fly* » à la réception des données. Le rendu final de l'application doit ressembler à ceci :



3.8 HTTPService avec paramètres

Dans cette application, nous n'allons pas charger un fichier statique mais faire appel à un service dont le résultat dépendra d'un paramètre.

Le service appelé est une page PHP, prenant un paramètre `id`.

*{SERVER URL}/ria/*getUserById.php

```
<?php
    header('Content-Type: application/xml');
    echo '<?xml version="1.0" encoding="UTF-8"?>';
    echo '<user id="'.$_GET["id"].'">';
    echo '<name>User "'.$_GET["id"].'"</name>';
    echo '<birthDate>1980-01-01</birthDate>';
    echo '</user>';
?>
```

Il s'agit de soumettre des valeurs de paramètres au service :

- En utilisant la propriété `request`

```
hseService.request = new Object();
hseService.request.name = txtSrc.text;
```

ou par un objet litéral :

```
hseService.request = {name:txtSrc.text};
```

- En utilisant le sous-élément `<request>`

```
<s:HTTPService
    id="hseService"
    url="http://localhost/ria/sayHello.php"
    …>
    <s:request>
        <name>{txtSrc.text}</name>
    </s:request>
</s:HTTPService>
```

Code source complet (avec paramètre par la propriété request)

```
<?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"
xmlns:components="components.*">
    <fx:Declarations>
        <s:HTTPService
            id="hseService"
            url="http://localhost/ria/getUserById.php"
            result="onResult(event)"
            fault="onFault(event)"
            showBusyCursor="true">
        </s:HTTPService>
    </fx:Declarations>

    <fx:Script>
        <![CDATA[
            import mx.controls.Alert;
            import mx.rpc.events.FaultEvent;
            import mx.rpc.events.ResultEvent;

            protected function cmdCallService_click(event:MouseEvent):void
            {
                // V1(a) - Propriété request
                hseService.request = new Object();
                hseService.request.id = txtSrc.text;
                hseService.send();
            }

            private function onResult(event:ResultEvent):void
            {
                var xml:XML = new XML(event.message.body);
                myUserRecord.userId.text = xml.@id;
                myUserRecord.username.text = xml.name;
                myUserRecord.birthDate.text = xml.birthDate;
            }

            private function onFault(event:FaultEvent):void
            {
                Alert.show('Broken service');
            }

        ]]>
    </fx:Script>

    <!-- Panel principal -->
    <s:Panel
        id="pnlClientServerSimple"
        title="Client/Serveur avec paramètres"
        verticalCenter="1"
        horizontalCenter="1">

        <s:layout>
            <s:VerticalLayout
                paddingBottom="10"
                paddingLeft="10"
                paddingRight="10"
                paddingTop="10"/>
        </s:layout>
```

```
        <!-- Zone de recherche -->
        <s:HGroup
            verticalAlign="middle">
            <s:Label
                text="User Id:"/>
            <s:TextInput
                id="txtSrc"/>
            <s:Button
                id="cmdCallService"
                label="Rechercher"
                enabled="{txtSrc.text!=''}"
                click="cmdCallService_click(event)"/>
        </s:HGroup>

        <!-- Colonnes -->
        <s:Group>
            <s:layout>
                <s:HorizontalLayout/>
            </s:layout>
            <s:Label
                width="25"
                text="Id"
                fontWeight="bold"
                fontSize="13"/>
            <s:Label
                width="150"
                text="User"
                fontWeight="bold"
                fontSize="13"/>
            <s:Label
                text="Birthdate"
                fontWeight="bold"
                fontSize="13"/>
        </s:Group>

        <!-- Zone d'affichage -->
        <components:UserRecord
            id="myUserRecord"/>

    </s:Panel>
</s:Application>
```

3.9 Exercices

GeoNames offre un ensemble de services géographiques permettant d'exploiter une base de toponymes avec couverture internationale. Il s'agit ici d'exploiter le service Postal Code Search et notamment son paramètre `placename_startsWith` permettant d'interroger la base pour trouver le nom d'un lieu (une ville). Exemple:

- Créer une RIA permettant de soumettre une telle requête avec un champ de saisie et un bouton d'envoi. Le flux XML résultat sera alors affiché dans une grille de donnée (colonnes = postalcode, name, countryCode, lat, lng).
- Créer une RIA sur la base de la précédente en enlevant le bouton, mais en invoquant la requête à chaque nouveau caractère saisi à partir de 3 caractères. Le résultat doit à présent alimenter un composant `<s:DropDownList>`.
- Créer la même RIA en utilisant le composant `<s:ComboBox>` permettant d'offrir à l'utilisateur une sorte d'auto-complétion à choix multiples.

```
API :
http://www.geonames.org/export/ws-overview.html

Exemple d'appel:
http://ws.geonames.org/postalCodeSearch?placename_startsWith=Frib&maxRows
=10
```

Google offre une API web pour soumettre des recherches depuis n'importe quelle application. L'interface web supporte la méthode GET et fournit un résultat au format JSON.

- Créer une RIA permettant d'initier une recherche sur la base d'un champ de saisie et d'un bouton d'envoi. Le résultat est alors représenté, notamment la propriété "title" au travers du composant `<s:RichText>`.

```
API :
http://code.google.com/apis/ajaxsearch/documentation/reference.html#_intr
o_fonje

as3corelib :
http://github.com/mikechambers/as3corelib
```

Remarque: il est nécessaire d'utiliser la librairie as3corelib et ce afin de décoder le flux JSON résultat.