	Formato FOR-DES-401.-Flex Builder V1.0.doc401 Flex Builder	Metodología: D.T.A.A.
		Coordinador: Marco García

HOJA DE CONTROL DE VERSIONES

Versión	Responsable del Cambio	Causa del Cambio	Nombre del Proyecto	Fecha del Cambio
1.0	Bruno Espinosa Diaz	Complementos de la información Generación del Documento	Tutorial Flex	19/02/08
1.1	Gabriela Ruiz Lechuga	Correcciones de datos	Tutorial Flex	20/02/08
1.2	Luis Barrita Arrieta	Correcciones de datos	Tutorial Flex	05/06/08



Formato FOR-DES-401.-Flex Builder
V1.0.doc401
Flex Builder

Metodología: D.T.A.A.

Coordinador: Marco
García

Índice

Página

1. DESCRIPCIÓN DE FLEX	4
1.1 INTRODUCCIÓN	4
1.1.1 ¿QUÉ ES FLEX?	4
1.1.2 RIA'S	4
1.1.3 VENTAJAS Y DESVENTAJAS:	4
2. INSTALACIÓN DE FLEX	5
2.1 INICIO	5
2.1.1 FIGURA 1	5
2.1.2 FIGURA 2	6
3. CREANDO UNA APLICACIÓN ESTÁTICA	7
3.1 CREAR UN PROYECTO EN ECLIPSE Y DESARROLLAR LA PRIMERA APLICACIÓN EN FLEX.	7
3.1.1 FIGURA 1	7
3.1.2 FIGURA 2	7
3.1.3 FIGURA 3	8
3.1.4 FIGURA 4	8
3.1.5 FIGURA 5	9
3.1.6 FIGURA 6	10
3.2 MODO DE DISEÑO.....	11
3.2.1 FIGURA 7	11
3.2.2 FIGURA 8	11
3.3 MODO DE CÓDIGO	12
3.3.1 FIGURA 9	12
4. INICIAR SERVIDOR	12
4.1 DESARROLLO	12
4.1.1 PROBAR APLICACIÓN.....	12
4.1.1 FIGURA 10	12
4.1.2 FIGURA 11	13
4.1.3 FIGURA 12	13
4.1.4 FIGURA 13	14
5. EJERCICIO 1.	14
5.1 DESARROLLO	14
5.1.1 SENTENCIAS DE CONTROL	14
5.1.2 FIGURA 14	14
5.1.3 FIGURA 15	15
5.2 COMBOBOX.....	17
5.2.2 DESARROLLO	17
5.3. DATAGRID	18
5.3.2 DESARROLLO	18



Formato FOR-DES-401.-Flex Builder
V1.0.doc401
Flex Builder

Metodología: D.T.A.A.
Coordinador: Marco
García

5.4 EJECUCION DE LA APLICACIÓN	19
5.4.1 DESARROLLO	19
5.4.1 FIGURA 16	19
5.4.2 FIGURA 17	19
5.4.3 FIGURA 18	19
6. GRÁFICOS ESTADÍSTICOS EN FLEX CON LINECHART Y TABNAVIGATOR	20
6.1. DESARROLLO	20
6.1.1 FIGURA 19	22
6.1.2 FIGURA20	23
6.1.3 FIGURA 21	24
6.2 LAS FUNCIONES.....	25
6.2.1 EJECUCIÓN DE “GRAFICA EJERCICIO2”	28
6.2.2 FIGURA 22	28
7. CREANDO UNA APLICACIÓN DINÁMICA	29
7.1 DESARROLLO	29
7.1.1 FIGURA 23	29
7.1.2 FIGURA 24	30
7.1.2 FIGURA 25	30
7.1.3 FIGURA 26	31
7.1.4 FIGURA 27	31
7.1.5 FIGURA 28	32
7.1.6 FIGURA 29	32
7.1.7 FIGURA 30	33
7.1.8 FIGURA 31	33
7.2 ACCESO A BASE DE DATOS	34
7.3 CONEXIÓN FLEX-JAVA	36



1. Descripción de Flex

1.1.Introducción

1.1.1 ¿Qué es Flex?

Flex es la solución de desarrollo de aplicaciones que Adobe pone a la mano, se destaca por ser muy completa y potente para crear y desarrollar aplicaciones de Internet sofisticadas (RIA, del inglés Rich Internet Applications) en la empresa y en la web. Permite a las empresas crear aplicaciones personalizadas y ricas en elementos multimedia que mejoran notablemente la experiencia del usuario y que están revolucionando la manera en que las personas interactúan con la web [Adobe].

1.1.2 RIA's

RIA, acrónimo de *Rich Internet Applications* (Aplicaciones de Internet Ricas o sofisticadas). Es un nuevo tipo de aplicación con más ventajas que las tradicionales aplicaciones Web. Surge como una combinación de las ventajas que ofrecen las aplicaciones Web y las aplicaciones tradicionales.

Las capacidades multimedia son totales gracias a que estos entornos tienen reproductores internos y no hace falta algún reproductor en la máquina cliente.


Hay muchas herramientas para la creación de entornos RIA. Entre estas podemos mencionar las plataformas Flash y Flex de Macromedia, AJAX, Open Lazlo (herramienta Open Source), Silverlight de Microsoft y JavaFx Script de Sun Microsystems. [Wikipedia]

1.1.3 Ventajas y desventajas:

Normalmente en las aplicaciones Web tradicionales, hay una recarga continua de páginas cada vez que el usuario pulsa sobre un enlace. De esta forma se produce un tráfico muy alto entre el cliente y el servidor, llegando muchas veces, a recargar la misma página con un mínimo cambio.

Otra de las desventajas de las tradicionales aplicaciones Web es la poca capacidad multimedia que posee. Para ver un vídeo tenemos que usar un programa externo.

En los entornos RIA, en cambio, no se producen recargas de página, ya que desde el principio se carga toda la aplicación y sólo se produce comunicación con el servidor cuando se necesitan datos externos como datos de una Base de Datos o de otros archivos externos, sin embargo, en la mayoría de los entornos RIA se necesita instalar un plugin o software adicional en la máquina cliente. [Wikipedia]

	Formato FOR-DES-401.-Flex Builder V1.0.doc401 Flex Builder	Metodología: D.T.A.A.
		Coordinador: Marco García

2. Instalación de Flex

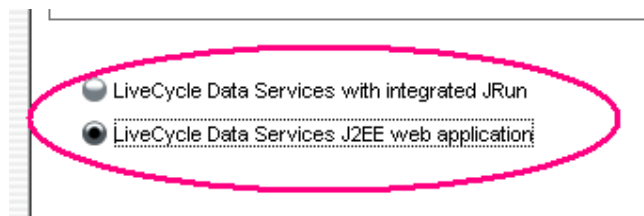
2.1.Inicio

Antes de instalar Flex es recomendable asegurarse de que tengamos instalado Eclipse 3.2.y Tomcat 5.5 o superior que será el servidor que usaremos para probar nuestras aplicaciones que desarrollemos.

2.2.Desarrollo

Primero instalamos el “Data Service 2 Express”, para ello ejecutamos el archivo “**lcds251-win.exe**” y aceptamos todas las preferencias por defecto. En las Opciones de Instalación seleccionamos “LiveCycle Data Services J2EE Web application” como se muestra en la figura y presionamos next e install:

2.2.1.Figura 1




Descripción: Selección de Opciones de Instalacion

Una vez realizado esto Ingresamos ala carpeta de instalación y cambiamos la extensión **.war** al archivo “**flex.war**” por **.rar**. Creamos una carpeta con el nombre de flex y dentro de ella se descomprime el archivo veremos que se han creado la carpeta **WEB_INF** y **META-INF** borramos *index.html*, una vez realizado esto copiaremos esta carpeta dentro de la carpeta **webapps** del **Tomcat**.

Finalmente instalaremos el plugin para Eclipse, ejecutamos el archivo “**FLXB_2.0_Win_WWE.exe**”, indicamos la ruta donde se almacenarán los archivos temporales de instalación, después comenzará el asistente de instalación.

- ☒ **Seleccionamos** “Flex Builder Plug-in and Flex SDK” y presionamos el botón “Next”
- ☒ **Aceptamos** la Introducción
- ☒ **Aceptamos** los términos de la licencia
- ☒ **Seleccionamos** una ruta para instalar el plugin y el JRE especial
- ☒ **Seleccionamos** la ruta donde se encuentra instalado eclipse
- ☒ **Indicamos** que deseamos instalar Flash Placer 9 para el navegador de nuestro equipo
- ☒ **Revisamos** las especificaciones de instalación



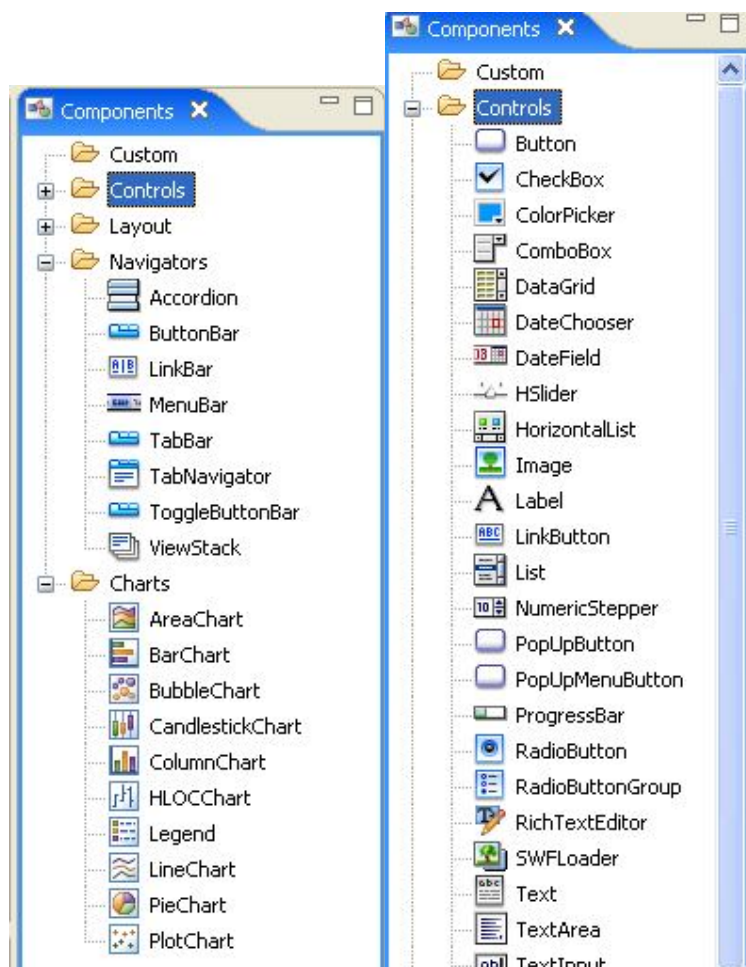
 **Al finalizar** se mostrará el mensaje que indica la instalación satisfactoria.




Todo el Software necesario se encuentra en el contenedor del área de capacitación.

Abra Eclipse 3.2 podrá observar que se encuentra una pestaña con los componentes para desarrollar las interfaces graficas.

2.2.2 Figura 2



Descripción: Componentes de diseño

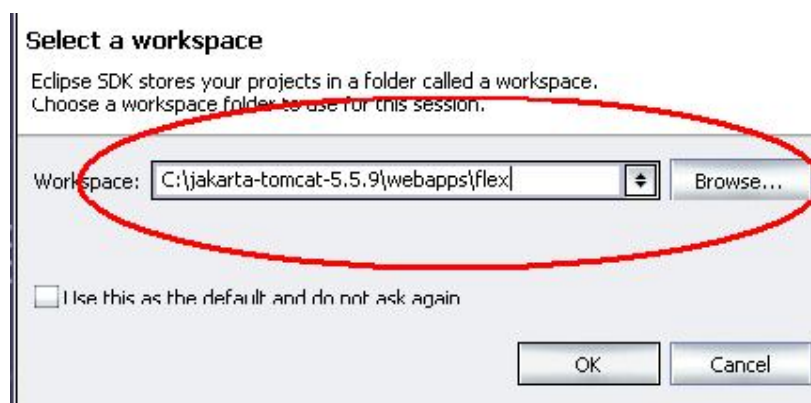
	Formato FOR-DES-401.-Flex Builder V1.0.doc401 Flex Builder	Metodología: D.T.A.A.
		Coordinador: Marco García

3. Creando una Aplicación Estática

3.1 Crear un Proyecto en Eclipse y desarrollar la primera aplicación en flex.

Para poder crear un proyecto Flex ya deberá tener descomprimido el archivo **flex.rar** en una carpeta de nombre **flex** dentro de la carpeta **weapps** del Tomcat 5.5, una vez verificado proceda a abrir Eclipse y elija la siguiente ruta de **workspace** para desarrollar: "C:\jakarta-tomcat-5.5.9\webapps\flex"

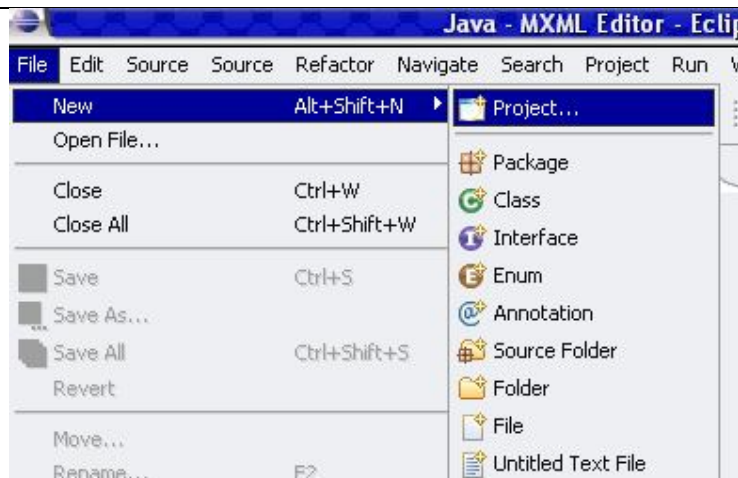
3.1.1 Figura 1



Descripción: Seleccionar workspace

Descripción: Primero Crearemos un proyecto dando clic en "File/New/Project..."

3.1.2 Figura 2



Descripción: Crear Nuevo Project

Aparecerá la siguiente ventana.

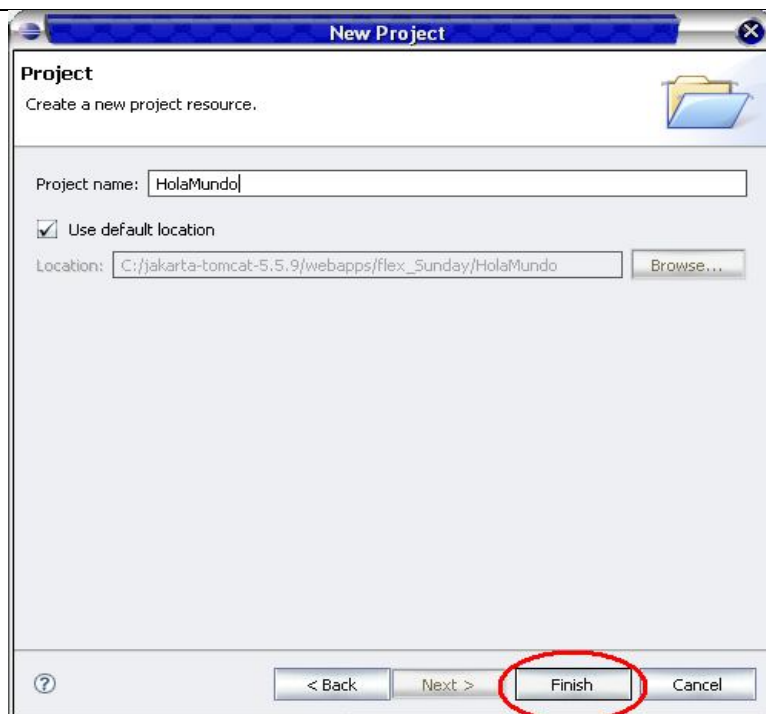
3.1.3 Figura 3



Descripción: Selección de un project

Seleccionamos “Project” presionamos “Next” y le asignamos un nombre y Presionamos Finish Como se muestra en la siguiente figura 4.

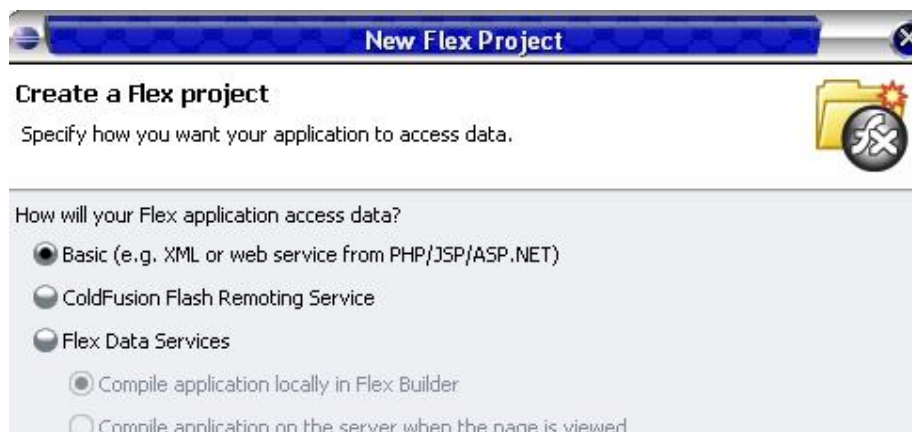
3.1.4 Figura 4




Descripción: Asignación de Nombre al project

A hora damos clic derecho sobre el proyecto recién creado y seleccionamos nuevo “Flex Project” de la carpeta “Flex” y presionamos el botón “Next”, entonces aparecerá la siguiente parte del asistente que se muestra en la figura 5.

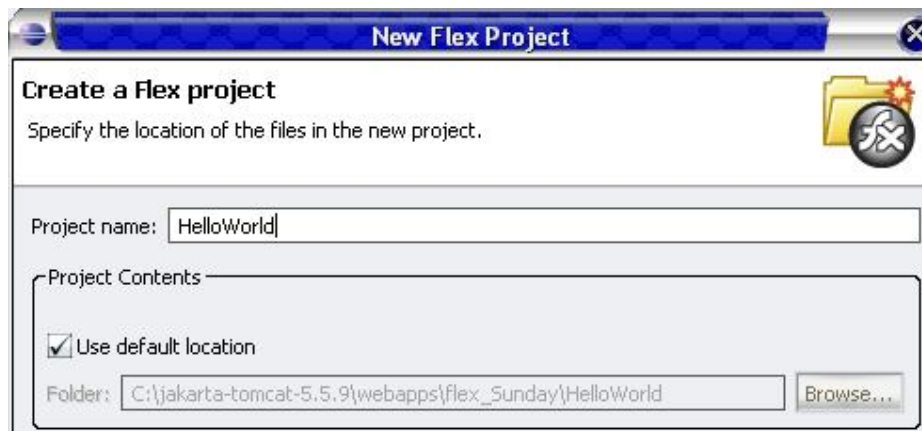
3.1.5 Figura 5



Descripción: Seleccionar el acceso a datos

	Formato FOR-DES-401.-Flex Builder V1.0.doc401 Flex Builder	Metodología: D.T.A.A.
		Coordinador: Marco García

Seleccionamos la opción “Basic” y presionamos el botón “Next”, entonces aparecerá la siguiente pantalla.

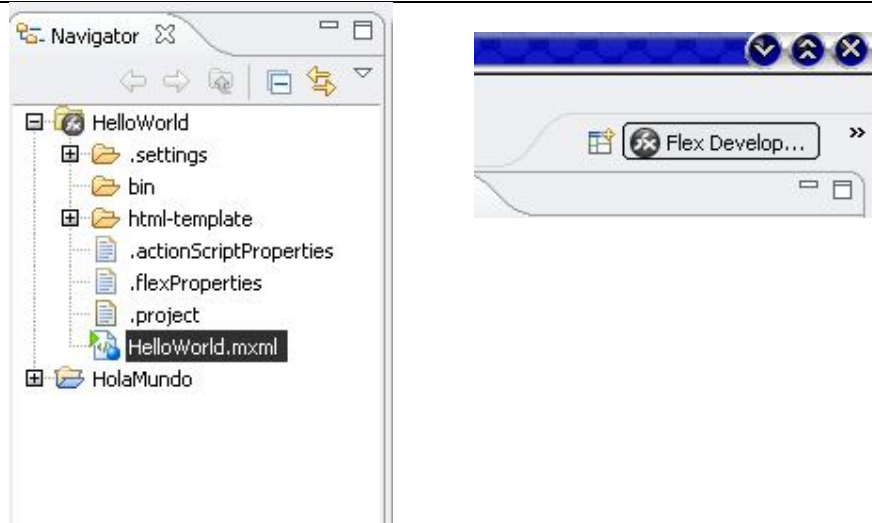


Descripción: Escribir el nombre del project

Aquí escribimos el nombre del proyecto y presionamos el botón “Finish”.

En este momento se debe crear el proyecto junto con una estructura de carpetas y una aplicación MXML que podemos visualizar en la ventana “Navegador” de Eclipse como se muestra en la figura 6. Para obtener el mejor provecho del IDE debemos asegurarnos que tenemos activada la perspectiva “Flex Development”.

3.1.6 Figura 6



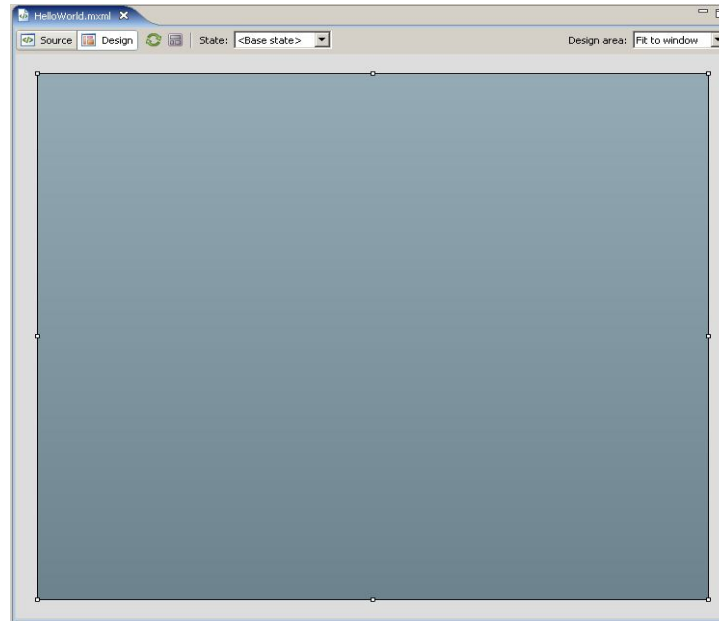
Descripción: Activación de la perspectiva

En la ventana de edición aparecen dos botones “Source” y “Design”, el primero sirve para poner el editor en modo de código y el segundo para ponerlo en modo diseño, la diferencia es que en modo “Source” tenemos que escribir directamente el código mientras que en modo “Design” se genera automáticamente cuando arrastramos y soltamos componentes a nuestro editor.

3.2 Modo de Diseño

El editor se presenta de la siguiente manera:

3.2.1 Figura 7



Descripción: Modo Design

Ahora, para insertar texto tomamos el componente “Label” de la carpeta “Controls” en la ventana “Components” (Fig. 3.1) y lo arrastramos a nuestro editor. En el momento en que soltamos el componente, aparecen sus propiedades en la ventana “Flex Properties”. En el campo “Text” Escribimos el texto deseado y presionamos Enter.

3.2.2 Figura 8



Descripción: Asignación de nombre ala etiqueta

3.3 Modo de Código

Cambiar al modo de código nuestro editor se ve de la siguiente manera:



3.3.1 Figura 9



Descripción: Modo Source

Para agregar un botón que despliegue un aviso tecleamos `<mx:Button label="Alert" click="mx.controls.Alert.show('Hola Mundo','Hola...') x="278" y="212"/>`

4. Iniciar Servidor

Antes de probar nuestra primer aplicación debemos arrancar el servidor Tomcat, para ello nos posicionamos en el directorio bin de Tomcat y ejecutamos el comando "C:\jakarta-tomcat-5.5.9\bin>startup".

4.1 Desarrollo

4.1.1 Probar Aplicación


Ingresamos <http://localhost:8080/manager/html> ya damos clic sobre la Trayectoria "/flex" como se muestra en la siguiente pantalla

4.1.1 Figura 10

Aplicaciones		
Trayectoria	Nombre a Mostrar	Ejecutándose
/	Welcome to Tomcat	true
/admin	Tomcat Administration Application	true
/balancer	Tomcat Simple Load Balancer Example App	true
/flex	LiveCycle Data Services	true
/flex_Sunday	LiveCycle Data Services	true

Descripción: Probar Aplicación

Ahora solo Ingresamos a la carpeta donde se encuentra nuestro proyecto flex y damos clic en el archivo "HelloWorld.mxml" como se muestra en la siguiente figura:

	Formato FOR-DES-401.-Flex Builder V1.0.doc401 Flex Builder	Metodología: D.T.A.A.
		Coordinador: Marco García

4.1.2 Figura 11

Listado de Directorio Para /HelloWorld/

Nombre de Archivo

.actionScriptProperties

.flexProperties

.project

.settings/

HelloWorld.mxml

bin/

html-template/

Descripción: Archivo a Ejecutar

A hora solo nos resta esperar a que se ejecute y deberá obtener una pantalla como la siguiente

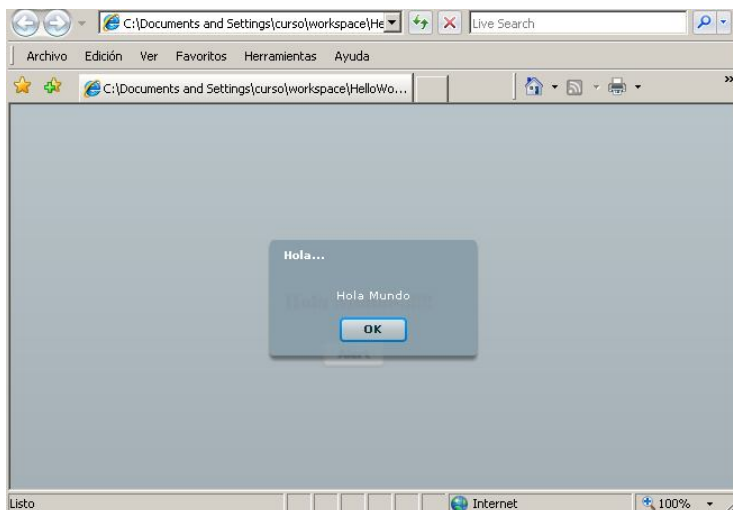
4.1.3 Figura 12



Descripción: Ejecución del proyecto

Y cuando pulsamos el botón “Alert” debe aparecer nuestro aviso así

4.1.4 Figura 13



Descripción: Evento al Presionar el botón



La descripción de estas pantallas hacen referencia al documento FOR-DES-501.- Pantallas, donde se especifica los requerimientos de la misma.

5. Ejercicio 1 (DIA 2)

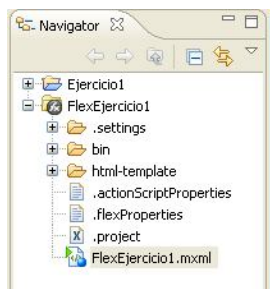
5.1 Desarrollo

5.1.1 Sentencias de Control


Este ejercicio hace Uso de sentencias de Control implementando un Combo que a su vez utiliza eventos para llenar una lista por medio de un ArrayCollection.

Antes que nada creamos un nuevo project llamado “Ejercicio1”, seguido de un proyecto Flex con nombre “FlexEjercicio1” como se muestra en la siguiente ventana.

5.1.2 Figura 14

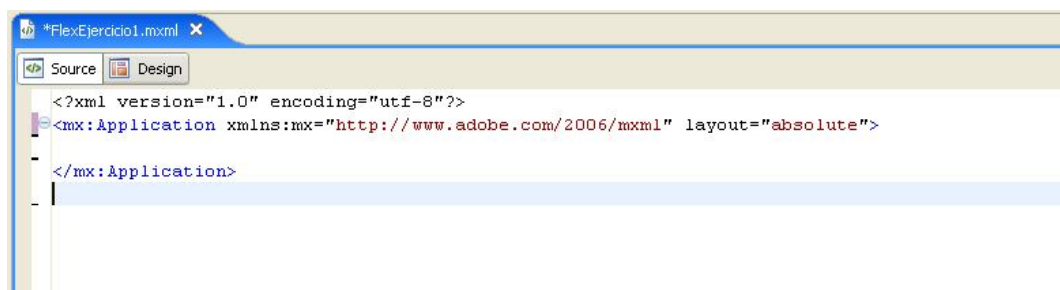


Descripción: Vista de Proyecto creado

	Formato FOR-DES-401.-Flex Builder V1.0.doc401 Flex Builder	Metodología: D.T.A.A.
		Coordinador: Marco García

Abra “**FlexEjercicio1.mxml**” se vera como se muestra en la siguiente figura15.

5.1.3 Figura 15



Descripción: Vista del mxml recién creado

El cuerpo de nuestro Programa ira dentro de estas Etiquetas de aplicación de flex ya que será la aplicación principal.

```
<mx:Application xmlns:mx=http://www.adobe.com/2006/mxml layout="absolute">
```

Cuerpo

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

Las acciones de **actionscript** en las aplicaciones Flex se escriben en una etiqueta (**<mx:Script>**) dentro de la etiqueta de la película (**<mx:Application>**) justo antes de las etiquetas de los componentes. Y a su vez a de ir incluida dentro de un nodo **CDATA** Ingrese las siguientes líneas en el archivo “FlexEjercicio1.mxml”.

FlexEjercicio1.mxml

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

  <mx:Script>
    <![CDATA[

//Aqui se escribirán las funciones

]]>

    <mx:ComboBox id="Combo" labelField="name" x="338" y="250"/>

    <mx:Panel x="500" y="250" title="Cinemas">
      <mx:DataGrid width="100%" height="100%" />
    </mx:Panel>

  </mx:Application>
```

Descripción: Etiquetas indispensables



Todas las funciones y operaciones deberan ir dentro de CDATA.

En el ActionScript empezamos entonces con importar la clase que nos permite trabajar con ArrayCollection, y luego, definir el objeto usando [Bindable] que será el encargado de enlazar los objetos como se muestra en las siguientes líneas remarcadas y que deberá ingresar en el archivo "FlexEjercicio1.mxml" y deberá lucir como se muestra en el siguiente cuadro de texto.

FlexEjercicio1.mxml

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

<mx:Script>
<![CDATA[

import mx.events.ListEvent;
import mx.collections.ArrayCollection;

[Bindable]
private var Cine : ArrayCollection;

[Bindable]
private var originalArrayCollection:ArrayCollection ;

]]>
</mx:Script>

<mx:ComboBox id="Combo" labelField="name" x="338" y="250"/>

<mx:Panel x="500" y="250" title="Cinemas">
<mx:DataGrid width="100%" height="100%" />
</mx:Panel>

</mx:Application>
```

Descripción: Etiqueta encargada de enlazar

A hora crearemos la funcion create() que sera la encargada de llenar el Combo. Ingrese las siguientes lineas en "**FlexEjercicio1.mxml**" dentro del nodo CDATA justo debajo de "originalArrayCollection".

```
private function create() : void {
Cine = new ArrayCollection();
var Item : Object;
Item = new Object();
Item.name = 'Lumiere';
Cine.addItem(Item);
Item = new Object();
Item.name = 'Cinemex';
Cine.addItem(Item);

Item = new Object();
Item.name = 'Teresa';
Cine.addItem(Item);
Combo.dataProvider= Cine;
}
```



5.2 ComboBox

5.2.2 Desarrollo

Dentro del componente **<mx:ComboBox>** ingrese la siguiente línea **creationComplete="create()"**, de esta forma el combo se inicializara para poder traer los datos que fueron almacenados en el **ArrayCollection Cines**, debiera lucir de esta forma.

FlexEjercicio1.mxml

```
<mx:ComboBox id="Combo" creationComplete="create()" labelField="name" />
```

Descripción: Inicialización del combo

Despues de la funcion "create()" ingrese la funcion **"getCineName(event: ListEvent)"** que sera la encargada de recibir los eventos del combo las lineas de codigo se muestra a continuacion:

FlexEjercicio1.mxml

```
private function getCineName( event: ListEvent) : void {  
  
    var id:int;  
  
    switch (Combo.selectedItem.name){  
  
        case 'Lumiere':  
            id=1;  
            break;  
  
        case 'Cinemex':  
            id=2;  
            break;  
  
        case 'Teresa':  
            id=3;  
            break;  
  
    }  
}
```

Descripción: Creación de la función getCineName()

A hora deberá ingresar la siguiente línea en el ComboBox **change="getCineName(event)"** quedando esa línea de código como se muestra a continuación:

FlexEjercicio1.mxml

```
<mx:ComboBox id="Combo" creationComplete="create()" labelField="name"  
change="getCineName(event)" x="338" y="250"/>
```



Descripción: Código Final del Combo



Esta linea `change="getCineName(event)"` enviara un evento, el cual sera cachado por la sentencia de control `switch` que se encuentra dentro de la funcion `getCineName(event: ListEvent)`. Para tomar una decisión y poder llenar la lista de las Películas dependiendo del **Cinema que se Seleccione**.

5.3. DataGrid

5.3.2 Desarrollo

- ☞ A Hora Agregaremos un **"DataGrid"** para desplegar los registros en una tabla.
- ☞ Dependiendo de la selección que hagamos en el ComboBox y que cambiara esta información cada vez que nosotros realicemos una nueva selección.
- ☞ El DataGrid deberá ir dentro de un **Panel**.
- ☞ El DataGrid contendra un **dataProvider** mediante el cual obtendremos los datos almacenados en el Array
- ☞ Ingrese la siguiente linea debajo del Combo.

FlexEjercicio1.mxml

```
<mx:Panel x="500" y="250" title="Cinemas">
    <mx:DataGrid width="100%" height="100%" dataProvider="{originalArrayCollection}" />
</mx:Panel>
```

Descripción: Creación del DataGrid

A hora deberá ingresar los siguientes if que dependiendo de la selección en el combo switch enviara el id de ese cine que se ha seleccionado estas líneas son:



```
if(id==1){
originalArrayCollection=new ArrayCollection([
{Pelicula:"Jack", Sala:"6", Funcion:"13:00-15:00"},
{Pelicula:"Rambo", Sala:"1", Funcion:"10:00-15:00"}
]);
}
if(id==2){
originalArrayCollection=new ArrayCollection([
{Pelicula:"Orfanato", Sala:"1", Funcion:"10:00-12:00"},
{Pelicula:"Sultanes", Sala:"2", Funcion:"13:00-17:00"},
{Pelicula:"Transformes",Sala:"3", Funcion:"10:00-12:00"},
{Pelicula:"Soy Leyenda", Sala:"4", Funcion:"9:00-16:00"}
]);
}
if(id==3){
originalArrayCollection=new ArrayCollection([
{Pelicula:"Cerrado", Sala:"Cerrado", Funcion:"Cerrado"}
]);
}
}
```

Descripción: Llena el Array

5.4 Ejecución de la aplicación

5.4.1 Desarrollo

Arranque el Servidor e Ingrese a Tomcat Manager, a continuación ingrese ala Trayectoria donde se encuentra su proyecto y ejecute el archivo con nombre "FlexEjercicio1.mxml" como se muestra en la figura

5.4.1 Figura 16

Nombre de Archivo

.actionScriptProperties

.flexProperties

.project

.settings/

FlexEjercicio1.mxml

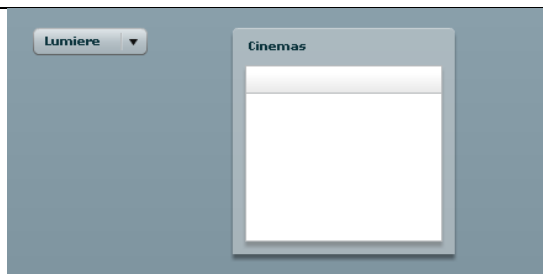
bin/

html-template/

Descripción: Archivo a Ejecutar

En ejecución se deberá Observar algo como en lo que se muestra en la figura 17.

5.4.2 Figura 17



Descripción: Ejecución del proyecto

Si realizamos una selección en el Combo deberá suceder lo que se muestra en la figura 18.

5.4.3 Figura 18



Descripción: Selección de un Cinemex



La descripción de estas pantallas hacen referencia al documento FOR-DES-501.- Pantallas, donde se especifica los requerimientos de la misma.



El código completo se puede encontrar dentro la carpeta de inicio de este tutorial en el archivo "FlexEjercicio1.rar".

6. Gráficos estadísticos en Flex con LineChart y TabNavigator (DIA 3)

6.1. Desarrollo

En esta parte veremos lo fácil que es el crear graficas en flex juntaremos la acción de estos componentes en una aplicación de ingeniería que tiene como propósito medir la resistividad de suelos o resistividad eléctrica de suelos. Aun no utilizamos conexión a base de datos seguimos haciendo una aplicación estática por lo que no tendrá ningún problema en que se ejecute.

Primero deberemos crear un nuevo Project con nombre "GraficaEjercicio2" seguido de un proyecto Flex al que llamaremos "FlexGraficaEjercicio2", esto nos creara un mxml automáticamente en el que comenzaremos a ingresar código.

Deberemos crear tres TextInput los cuales tendrán la siguiente nomenclatura: para los de norte a sur los llamamos rns_1, rns_2, rns_2. lo análogo para los de este a oeste: rwe_1, rwe_2, rwe_3. El código para estos TextInput Se muestra a continuación.



GraficaEjercicio2.mxml

```
<mx:Panel x="10" y="10" width="496" height="296" layout="absolute" id="mypanel0" title="Ingreso de Datos">
<mx:TextInput x="224.25" y="19" width="58" id="rns_1" textAlign="center"/>
<mx:TextInput x="224.25" y="49" width="58" id="rns_2" textAlign="center"/>
<mx:TextInput x="224.25" y="79" width="58" id="rns_3" textAlign="center"/>
<mx:TextInput x="320.25" y="19" width="58" id="rwe_1" textAlign="center"/>
<mx:TextInput x="320.25" y="49" width="58" id="rwe_2" textAlign="center"/>
<mx:TextInput x="320.25" y="79" width="58" id="rwe_3" textAlign="center"/>
</mx:Panel>
```

Descripción: TextInput

Vamos a construir de una vez el navegador con pestañas TabNavigator, un sistema muy sencillo y eficiente para organizar información en espacios pequeños. La aplicación tiene tres áreas para mostrar al usuario que son las siguientes:

- ☒ **Pestaña** 'Valores de Resistencia'.
- ☒ **Pestaña** 'Grilla de Cálculos'.
- ☒ **Pestaña** 'Perfiles de Resistividad'.

Usaremos el componente '**Canvas**' que se encuentra en **Layout**, este componente contendrá todo lo visible en cualquiera de las 3 áreas a mostrar al usuario Por lo que requerimos tres Canvas, de esta forma:

GraficaEjercicio2.mxml

```
<mx:TabNavigator width="518" height="349" id="bigcontainer" selectedIndex="0" horizontalCenter="8" top="24">

  <mx:Canvas label="Valores de Resistencia" width="100%" height="100%">
    <!-- Aquí va el contenido de la interface 1 -->
  </mx:Canvas>

  <mx:Canvas label="Grilla de Calculos" width="100%" height="100%">
    <!-- Aquí va el contenido de la interface 2 -->
  </mx:Canvas>

  <mx:Canvas label="Perfiles de Resistividad" width="100%" height="100%">
    <!-- Aquí va el contenido de la interface 3 -->
  </mx:Canvas>

</mx:TabNavigator>
```

Descripción: Canvas para Formar las Pestañas



Formato FOR-DES-401.-Flex Builder
V1.0.doc401
Flex Builder

Metodología: D.T.A.A.

Coordinador: Marco
García

En cada uno de estos estados vamos a incluir los componentes que necesitamos. Se supone que al hacer clic en la pestaña correspondiente se mostrarán los componentes contenidos en el Canvas actual del TabNavigator. el componente se encarga de todo.

Entonces, se detallara rápidamente los componentes que hay en los 3 Canvas del TabNavigator:

Canvas 'Valores de Resistencia':

- Un contenedor Panel (solo para fines estéticos)
- 6 TextInput
- 5 Label para indicar cierta información

El código para Valores de Resistencia se muestra a continuación:

GraficaEjercicio2.mxml

```
<mx:Canvas label="Valores de Resistencia" width="100%" height="100%">
<mx:Panel x="10" y="10" width="496" height="296" layout="absolute" id="mypanel0" title="Ingreso de Datos">
<mx:TextInput x="224.25" y="19" width="58" id="rns_1" textAlign="center"/>
<mx:TextInput x="224.25" y="49" width="58" id="rns_2" textAlign="center"/>
<mx:TextInput x="224.25" y="79" width="58" id="rns_3" textAlign="center"/>
<mx:TextInput x="320.25" y="19" width="58" id="rwe_1" textAlign="center"/>
<mx:TextInput x="320.25" y="49" width="58" id="rwe_2" textAlign="center"/>
<mx:TextInput x="320.25" y="79" width="58" id="rwe_3" textAlign="center"/>
<mx:Label x="155.75" y="20" text="1" textAlign="center"/>
<mx:Label x="155.75" y="50" text="2" textAlign="center"/>
<mx:Label x="155.75" y="80" text="3" textAlign="center"/>
<mx:Label x="207.25" y="1" text="Resistencia N-S" fontWeight="bold"/>
<mx:Label x="305.25" y="1" text="Resistencia W-E" fontWeight="bold"/>
</mx:Panel>
</mx:Canvas>
```

Descripción: Código de la Canva Valores de Resistencia

Esta Canva se deberá visualizar de la siguiente forma:

6.1.1 Figura 19

The screenshot shows a software interface with three tabs: 'Valores de Resistencia', 'Grilla de Calculos', and 'Perfiles de Resistividad'. The 'Valores de Resistencia' tab is active, displaying a window titled 'Ingreso de Datos'. Inside this window, there are two columns of input fields. The first column is labeled 'Resistencia N-S' and the second is labeled 'Resistencia W-E'. There are three rows of input fields, numbered 1, 2, and 3 on the left. The first row has a value '1' entered in the 'Resistencia W-E' field.



Descripción: Visualización de la primera Canva

Canvas 'Grilla de Cálculos':

- Un Panel
- Un Datagrid con 7 columnas

El código que deberá ir dentro de esta Canva es el siguiente:

GraficaEjercicio2.mxml

```
<mx:Canvas label="Grilla de Calculos" width="100%" height="100%">
<mx:DataGrid x="6" y="10" width="464" height="224" id="myGrid" dataProvider="{mydata}">

<mx:columns>

<mx:DataGridColumn headerText="Prof.(m)" dataField="profundidad" headerStyleName="center" textAlign="center"/>
<mx:DataGridColumn headerText="Sep.(m)" dataField="separacion" textAlign="center"/>
<mx:DataGridColumn headerText="R-NS" dataField="resistencia_ns" headerStyleName="center" textAlign="center"/>
<mx:DataGridColumn headerText="R-EO" dataField="resistencia_we" textAlign="center"/>
<mx:DataGridColumn headerText="p-NS" dataField="resistividad_ns" textAlign="center"/>
<mx:DataGridColumn headerText="p-EO" dataField="resistividad_we" textAlign="center"/>
<mx:DataGridColumn headerText="Prom." dataField="promedio" textAlign="center"/>
</mx:columns>

</mx:DataGrid>
</mx:Canvas>
```

Descripción: Código de la Canva Grilla de Calculos



Preste atención a los 'dataField'. Cada uno de estos campos va a estar enlazado a cada uno de los componentes de la variable 'mydata', es decir, en la estructura de mydata, vamos a encontrar a 'profundidad', 'separación', 'resistencia'. El resto de parámetros son solo ajustes de visualización.

Se deberá Visualizar lo siguiente:

6.1.2 Figura20



Formato FOR-DES-401.-Flex Builder
V1.0.doc401
Flex Builder

Metodología: D.T.A.A.

Coordinador: Marco
García

Valores de Resistencia

Grilla de Calculos

Perfiles de Resistividad

Prof.(m)	Sep.(m)	R-NS	R-EO	p-NS	p-EO	Prom.
0.75	1			0	0	0
1.5	2			0	0	0
2.25	3			0	0	0

Descripción: Visualización de la Segunda Canva

Canvas 'Perfiles de Resistividad'

- Un Panel
- Un componente LineChart para graficación
- Un componente Legend para acompañar al LineChart

El código que deberá contener es el siguiente:

GraficaEjercicio2.mxml

```
<mx:Canvas label="Perfiles de Resistividad" width="100%" height="100%">
  <mx:Panel x="10" y="10" width="496" height="296" layout="absolute" id="mypanel2" title="Grafico de Perfiles de Resistividad">
    <mx:LineChart x="6" y="2" id="mychart" height="220" width="464" dataProvider="{mydata}" showDataTips="true">
      <mx:horizontalAxis>
        <mx:CategoryAxis categoryField="profundidad"/>
      </mx:horizontalAxis>
      <mx:series>
        <mx:LineSeries displayName="Resistividad Promedio" yField="promedio" form="curve"/>
        <mx:LineSeries displayName="Resistividad N-S" yField="resistividad_ns" form="curve"/>
        <mx:LineSeries displayName="Resistividad W-E" yField="resistividad_we" form="curve"/>
      </mx:series>
    </mx:LineChart>
    <mx:Legend dataProvider="{mychart}" x="32" y="225" direction="horizontal"/>
  </mx:Panel>
</mx:Canvas>
```

Descripción: Código de la Canva Perfiles de Resistividad



La descripción de estas pantallas hacen referencia al documento FOR-DES-501.- Pantallas, donde se especifica los requerimientos de la misma.

Este código dará lugar a esta visualización en el momento de ejecutarlo:

6.1.3 Figura 21



Descripción: Visualización de la Tercera Canva

Los valores serán calculados en el momento en que se presionan las pestañas del TabNavigator a través del evento change del mismo; este evento change aparece en la línea donde están las propiedades del TabNav la cual es la siguiente:

GraficaEjercicio2.mxml

```
<mx:TabNavigator width="518" height="349" id="bigcontainer" selectedIndex="0" horizontalCenter="8" top="24"
change="calculate()">
```

Descripción: Llama a Calculate()

La función 'calculate()' y el código para esta función es el siguiente:

GraficaEjercicio2.mxml

```
private function calculate():void {
    mydata.removeAll();
    setNewItem(0.75, 1, rns_1.text, rwe_1.text);
    setNewItem(1.5, 2, rns_2.text, rwe_2.text);
    setNewItem(2.25, 3, rns_3.text, rwe_3.text);
}
```

Descripción: Función Calculate()



En el ActionScript empezamos entonces con importar la clase que nos permite trabajar con ArrayCollection, y luego, definir el objeto usando [Bindable]. Junto con esto, hay una función que llamada init() que deberá correr apenas la aplicación sea cargada (mydata se deberá inicializar una vez). Para esto introducimos el evento creationComplete="init()" en el <mx:Application > quedando de la siguiente forma:

GraficaEjercicio2.mxml

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

Descripción: Inicializa mydata

El código para la función init() es el siguiente:

GraficaEjercicio2.mxml

```
<mx:Script>
<![CDATA[
    import mx.collections.ArrayCollection;

    [Bindable]
    private var mydata:ArrayCollection;
    private function init():void {
        mydata = new ArrayCollection();
    }
}]>
</mx:Script>
```

Descripción: Función init()

6.2 Las funciones

Tenemos en total 4 pequeñas funciones que nos permiten hacer lo que necesitamos, y son:

- init()
- calculate()
- setNewItem()
- myround()

Vamos una por una:

init()

Nos permite inicializar el proveedor de datos del sistema (mydata) en el momento de la carga de la aplicación.

calculate()

No es precisamente la función que ejecuta los cálculos, pero se encarga de llamar 8 veces a la función que si lo hace (setNewItem()) y también, antes de eso, limpia el proveedor de datos con removeAll().



setNewItem()

Esta si es la que ejecuta los cálculos. A partir de las resistencias enviadas desde calculate(), se determinan las resistividades a través de una fórmula sencilla ($2 \cdot \pi \cdot \text{SeparacionElectrodos} \cdot \text{Resistencia}$). Como sólo queremos dos decimales luego de la coma, se hizo una función llamada round() que se encarga de eso.

round()

Redondea el número que se le envía a dos decimales.

El código de estas funciones se muestra completo a continuación:

GraficaEjercicio2.mxml

```
<mx:Script>
  <![CDATA[
import mx.collections.ArrayCollection;

[Bindable]
private var mydata:ArrayCollection;
private var pns:Number = new Number();
private var pwe:Number = new Number();
private var prom:Number = new Number();
private function init():void {
mydata = new ArrayCollection();
}
private function calculate():void {
mydata.removeAll();
setNewItem(0.75, 1, rns_1.text, rwe_1.text);
setNewItem(1.5, 2, rns_2.text, rwe_2.text);
setNewItem(2.25, 3, rns_3.text, rwe_3.text);
}
private function setNewItem(prf:Number, sep:Number, rns:String, rwe:String):void {
pns = myround(2*(Math.PI)*sep*(new Number(rns)));
pwe = myround(2*(Math.PI)*sep*(new Number(rwe)));
prom = myround((pns + pwe)/2);
mydata.addItem({profundidad: prf, separacion: sep, resistencia_ns: rns,
resistencia_we: rwe, resistividad_ns: pns, resistividad_we: pwe, promedio: prom});
}
private function myround(numToRound:Number):Number {
numToRound = (Math.round(numToRound*100))/100;
return numToRound;
}
  ]>
</mx:Script>
```

Descripción: Funciones



Y para finalizar, el plato fuerte, el LineChart.

Hay varios componentes Chart en el IDE de Flex, Aquí se usara el LineChart porque es el que más se ajustaba al objetivo. Hay algunas cosas en la configuración que son importantes mencionar:

- ☒ **LineChart** requiere de un proveedor de datos. En este caso lo vinculamos a mydata quien suministra los datos en toda la aplicación. Se puede observar en la línea del LineChart:

GraficaEjercicio2.mxml

```
<mx:LineChart x="6" y="2" id="mychart" height="220" width="464" dataProvider="{mydata}" showDataTips="true">
```

Descripción: Vinculación a mydata por medio de dataProvider

- ☒ Al ajustar showDataTips en 'true' como se puede ver arriba, podemos obtener los los puntos de muestreo ingresados.
- ☒ Es bueno detallarle al LineChart quien va a a proveer los datos del eje horizontal. Esto se hace con mx:horizontalAxis como se muestra en la siguiente linea:

GraficaEjercicio2.mxml

```
<mx:horizontalAxis>  
  <mx:CategoryAxis categoryField="profundidad" />  
</mx:horizontalAxis>
```

Descripción: Proveedor de los datos del eje horizontal

- ☒ El campo '**profundidad**' (de la estructura **mydata**) es el que queremos que proporcione los datos del eje horizontal. Si puede probar con xField en los **LineSeries** (explicados abajo) tal vez se obtendrá un resultado que igual, funciona.
- ☒ Como son 3 gráficas las que vamos a incluir en el **LineChart**, necesitamos incluir tres elementos **<mx:LineSeries />**.
- ☒ Cada uno de ellos debe ir enlazado a los campos que corresponden a las resistividades calculadas (y al promedio por supuesto) a través de los campos de **mydata**: promedio, resistividad_ns y resistividad_we. Recordemos que en la función **setNewItem()** se construyó así la estructura, debemos usar exactamente los mismos nombres proporcionados a **mydata**.
- ☒ Debajo del **LineChart** está un componente llamado **Legend**. Observa aquí que el proveedor de datos de este componente es precisamente el **LineChart** (que en este caso se llama '**mychart**').



El código completo se puede encontrar dentro la carpeta de inicio de este tutorial en el archivo "GraficaEjercicio2.rar".

El código correspondiente a Perfiles de Resistividad que es el encargado de graficar y donde se encuentra el componente **LineChart** Se muestra a continuación.

GraficaEjercicio2.mxml

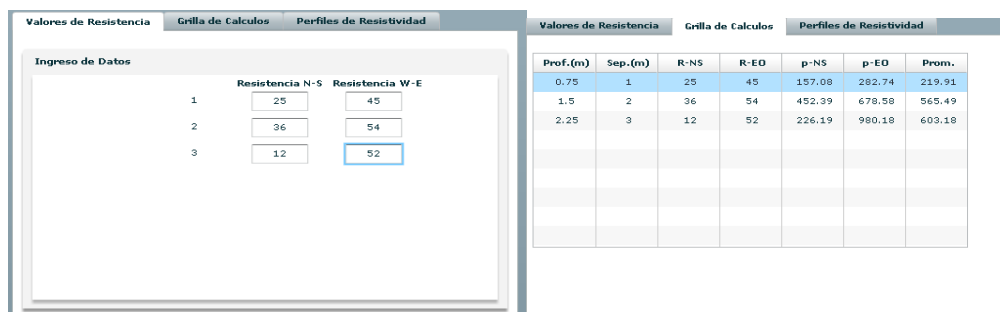
```
<mx:Canvas label="Perfiles de Resistividad" width="100%" height="100%">
<mx:Panel x="10" y="10" width="496" height="296" layout="absolute" id="mypanel2" title="Grafico de Perfiles de Resistividad">
<mx:LineChart x="6" y="2" id="mychart" height="220" width="464" dataProvider="{mydata}" showDataTips="true">
<mx:horizontalAxis>
<mx:CategoryAxis categoryField="profundidad"/>
</mx:horizontalAxis>
<mx:series>
<mx:LineSeries displayName="Resistividad Promedio" yField="promedio" form="curve"/>
<mx:LineSeries displayName="Resistividad N-S" yField="resistividad_ns" form="curve"/>
<mx:LineSeries displayName="Resistividad W-E" yField="resistividad_we" form="curve"/>
</mx:series>
</mx:LineChart>
<mx:Legend dataProvider="{mychart}" x="32" y="225" direction="horizontal"/>
</mx:Panel>
</mx:Canvas>
```

Descripción: Proveedor de los datos del eje horizontal

6.2.1 Ejecución de "GraficaEjercicio2"

El proyecto en ejecución deberá Visualizarse de la siguiente forma en sus tres Componentes del TabNavigator:

6.2.2 Figura 22





Descripción: Ejecución del proyecto terminado



La descripción de estas pantallas hacen referencia al documento FOR-DES-501.- Pantallas, donde se especifica los requerimientos de la misma.

7. Creando una aplicación dinámica (DIA 4)

7.1 Desarrollo

Existen tres maneras de obtener información dinámica a través de llamadas a procedimientos remotos (RCP), mediante Objetos Remotos, Web Services o http Services. En esta ocasión probaremos con objetos remotos de java.

Antes que nada creamos una carpeta en el **Tomcat** en la carpeta **webapps** con nombre **FlexRemote** y dentro descomprimos el archivo flex.rar que se menciona al principio del tutorial para poder trabajar con Tomcat y donde Trabajaremos.

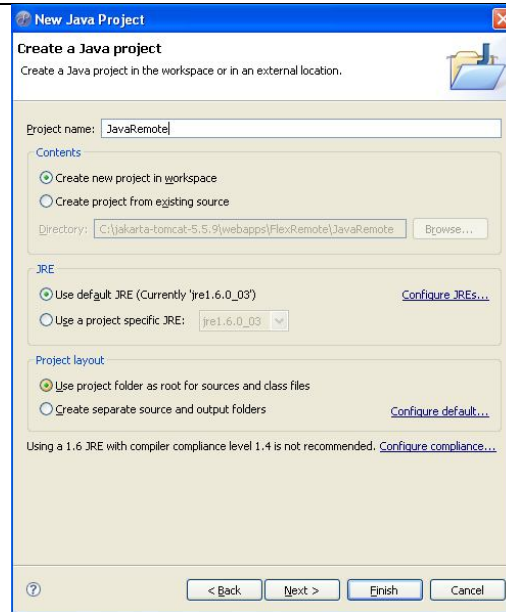
Después Abrimos Eclipse 3.2 y elegimos el **workspace** donde creamos nuestra carpeta **FlexRemote**. Una vez hecho esto Crearemos un nuevo project llamado "**RemoteObj**".

A hora crearemos un proyecto Java con el nombre **JavaRemote** el cual deberá ser linkeado para que las clases se generen automáticamente en la siguiente ruta
C:\jakarta-tomcat-5.5.9\webapps\FlexRemote\WEB-INF\classes

Para esto siga los siguientes Pasos:

Cree el proyecto Java y asígnele el nombre **JavaRemote** como se muestra en la figura 23:

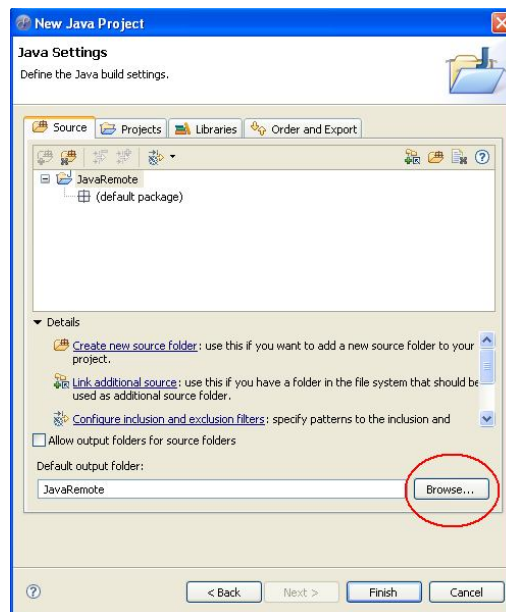
7.1.1 Figura 23



Descripción: Creación del proyectoJava

Haga clic en Next y en la Siguiente visualización se vera como en la siguiente figura.

7.1.2 Figura 24



Descripción: Selección de Carpeta

Presione el Boton **Browse** como se muestra en la figura24 y se abrirá esta otra ventana:

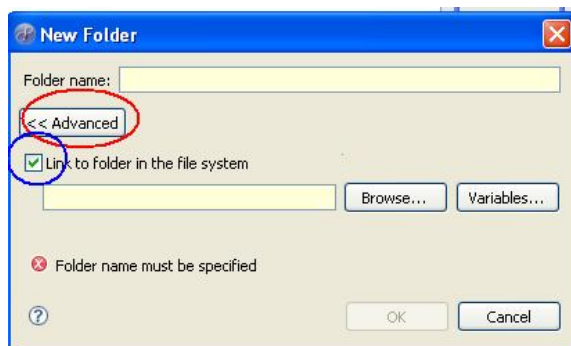
7.1.2 Figura 25



Descripción: Elegir create New Folder

Presione Create New Folder como se aprecia en la figura25 y aparecerá esta otra ventana. Donde Presionara Advanced y marcara la casilla Link to folder in the file system.

7.1.3 Figura 26

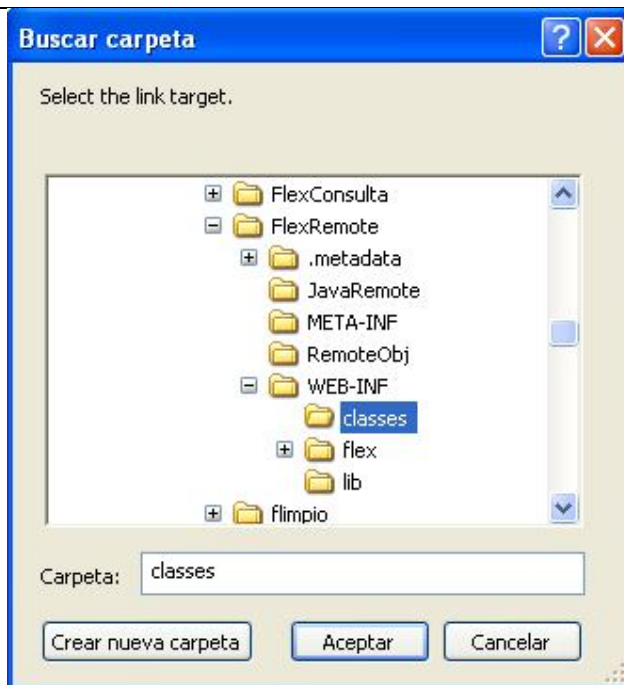


Descripción: Linkear a una carpeta externa

A hora presione Browse y elija la ruta de la carpeta donde se encuentra nuestro workspace y acontinuacion la carpeta WEB-INF/clases. Que será la carpeta ala cual nuestro proyecto flex mediante una configuración en el flex-config.xml se conectara a nuestros objetos java.

Seleccione esta ruta y deberá aparecer una ventana como la siguiente:

7.1.4 Figura 27

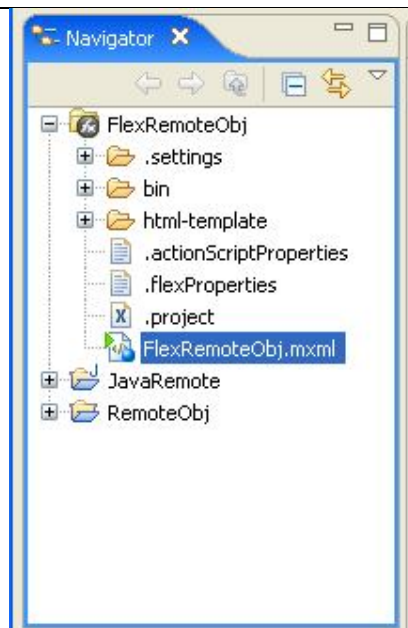


Descripción: Selección de la carpeta donde se depositaran los .class

Presione aceptar y Ok a todas las ventanas y finalmente finish.como test cree una clase y revise la siguiente ruta **C:\jakarta-tomcat-5.5.9\webapps\FlexRemote\WEB-INF\classes** y podrá observar aue se a creado el .class correspondiente al .java que creo.

A hora cree un proyecto Flex con nombre "**FlexRemoteObj**" como se muestra en la siguiente ventana.

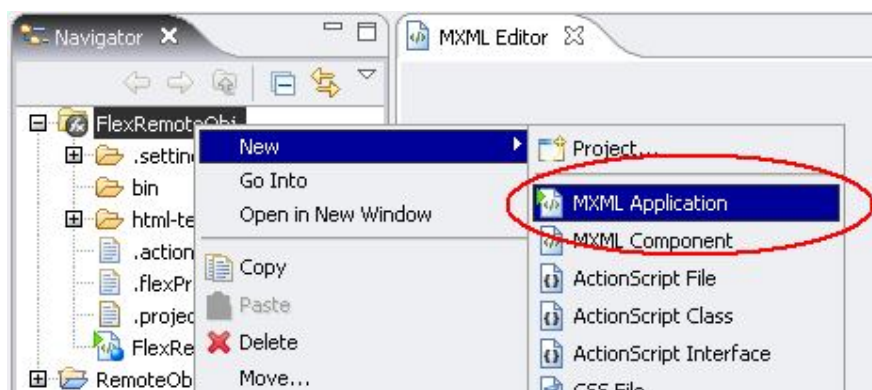
7.1.5 Figura 28



Descripción :Proyectos creados

Crear un MXML Application llamado “consulta”

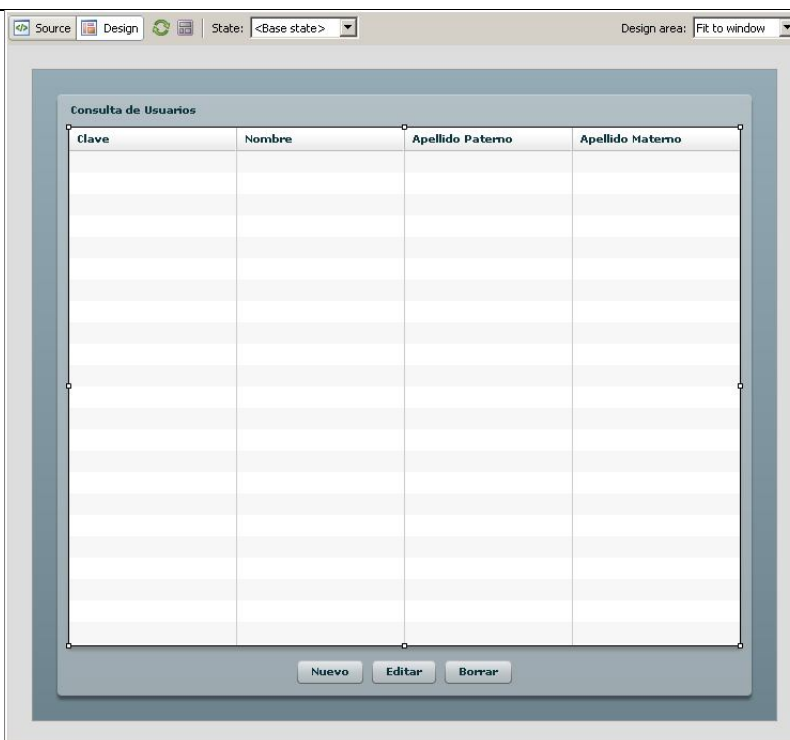
7.1.6 Figura 29



Descripción :creación consulta.mxml

A Hora insertamos un “dataGrid” para desplegar los registros de una tabla y agregamos tres botones para insertar, modificar y eliminar registros. Al final nuestro diseño debe ser similar al de la Fig.29

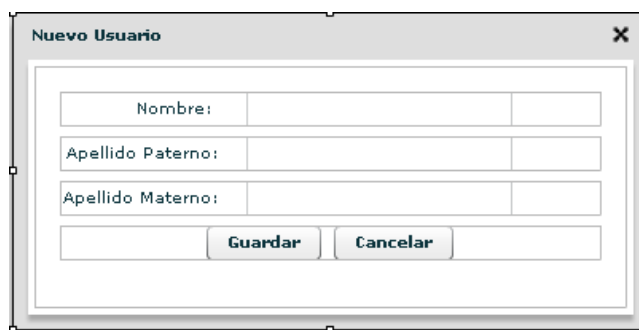
7.1.7 Figura 30



Descripción: creación del dataGrid

Posteriormente creamos un componente “TitleWindow” donde pondremos un formulario con tres textInput: Nombre, Apellido Paterno y Apellido Materno, además dos botones: Aceptar y Cancelar, quedando nuestro componente de la siguiente manera:

7.1.8 Figura 31



Descripción: creación del componente TitleWindow

Finalmente creamos otro componente igual para edición.

7.2 Acceso a Base de Datos

Ahora vamos a crear las clases en java que manejarán los accesos a la base de datos.



Formato FOR-DES-401.-Flex Builder
V1.0.doc401
Flex Builder

Metodología: D.T.A.A.

Coordinador: Marco
García

Comenzamos con una clase con el patrón de diseño “singleton” que creará conexiones a la base de datos esta clase se deberá llamar ConnFactory podrá encontrarla en el zip que se indica al final al igual que podrá encontrar todas las clases y mxml que se necesita para este proyecto al igual que la configuración del remoting-config.xml.

Para transportar los datos a las diferentes instancias de la aplicación creamos una clase con el patrón “Data Transfer Object” o “DTO”

```
public int getIdUsr() {  
    return idUsr;  
}  
  
public void setIdUsr(int idUsr) {  
    this.idUsr = idUsr;  
}  
  
public String getMaterno() {  
    return materno;  
}  
  
public void setMaterno(String materno) {  
    this.materno = materno;  
}  
  
public String getNombre() {  
    return nombre;  
}  
  
public void setNombre(String nombre) {  
    this.nombre = nombre;  
}  
  
public String getPaterno() {  
    return paterno;  
}  
  
public void setPaterno(String paterno) {  
    this.paterno = paterno;  
}  
  
public static long getSerialVersionUID() {  
    return serialVersionUID;  
}
```

Después continuamos creando una clase con el patrón “DAO” (Data Access Object) que se encargará de los accesos a la base de datos.



```
public List getUsuariosDAO(){
    List lista=new ArrayList();
    try{
        stm=conn.createStatement();
        String consulta="SELECT * FROM usuarios";
        rs=stm.executeQuery(consulta);
        while(rs.next()){
            ConsultaUsrDTO dto=new
ConsultaUsrDTO(rs.getInt(4),rs.getString(1),rs.getString(2),rs.getString(3));
            lista.add(dto);
        }
    }catch(Exception e){
        e.printStackTrace();
    }finally{
        closeConnection();
    }
    return lista;
}

public boolean saveUsrDTO(ConsultaUsrDTO dto){
    try{
        PreparedStatement ps = conn.prepareStatement("INSERT INTO usuarios (nombre,apaterno,amaterno)
VALUES(?, ?, ?)");
        ps.setString(1, dto.getNombre());
        ps.setString(2, dto.getPaterno());
        ps.setString(3, dto.getMaterno());
        ps.executeUpdate();
    }catch(Exception e){
        e.printStackTrace();
        return false;
    }finally{
        closeConnection();
    }
    return true;
}

public boolean deleteUsrDTO(int id){
    try{
        PreparedStatement ps = conn.prepareStatement("DELETE FROM usuarios WHERE idUsr=?");
        ps.setInt(1, id);
        ps.executeUpdate();
        return true;
    }catch(Exception e){
        e.printStackTrace();
        return false;
    }finally{
        closeConnection();
    }
}

public boolean updateUsrDTO(ConsultaUsrDTO dto){
    try{
        PreparedStatement ps = conn.prepareStatement("UPDATE usuarios SET
nombre=?,apaterno=?,amaterno=? WHERE idUsr=?");
        ps.setString(1, dto.getNombre());
        ps.setString(2, dto.getPaterno());
        ps.setString(3, dto.getMaterno());
        ps.setInt(4, dto.getIdUsr());
        ps.executeUpdate();
    }catch(Exception e){
        e.printStackTrace();
        return false;
    }finally{
        closeConnection();
    }
    return true;
}
}
```

Para la lógica de negocio creamos una clase "Bussines Object" (BO)



```
package com.ix.e.inegocios.persistencia.bo;

import java.util.List;
import dao.ConsultaUsrDAO;
import .dto.ConsultaUsrDTO;

public class ConsultaUsrBO {

    public ConsultaUsrBO(){

    }

    public List getUsuarios(){
        ConsultaUsrDAO dao=new ConsultaUsrDAO();
        List lista=dao.getUsuariosDAO();
        return lista;
    }

    public boolean saveUsr(ConsultaUsrDTO dto){
        ConsultaUsrDAO dao=new ConsultaUsrDAO();
        try{
            return dao.saveUsrDTO(dto);
        }catch(Exception e){
            return false;
        }
    }

    public boolean deleteUsr(String id){
        ConsultaUsrDAO dao=new ConsultaUsrDAO();
        try{
            return dao.deleteUsrDTO(Integer.parseInt(id));
        }catch(Exception e){
            return false;
        }
    }

    public boolean updateUsr(ConsultaUsrDTO dto){
        ConsultaUsrDAO dao=new ConsultaUsrDAO();
        try{
            return dao.updateUsrDTO(dto);
        }catch(Exception e){
            return false;
        }
    }
}
```

7.3 Conexión Flex-Java

Para comunicar una aplicación Flex con Java mediante Objetos Remotos necesitamos configurar el archivo "remoting-config.xml" dentro del directorio donde descomprimos el flex.ra buscamos esta dirección "\\WEB-INF\\flex\\remoting-config.xml". Aquí agregamos el siguiente código:

```
<destination id="consultaUsr">
    <properties>
        <source>bo.ConsultaUsrBO</source>
        <scope>application</scope>
    </properties>
</destination>
```



Posteriormente creamos una clase **ActionScript** que contenga únicamente las propiedades de nuestro DTO **ISystemManager** sera el encargado de llamar ala clase Remota que declaramos en en el remoting-config.xml y **[Bindable]** sera el encargado de Enlazarse con los objetos remotos Java esta clase tiene el nombre de **UsrDTO.as**.

```
package
{
    import mx.managers.ISystemManager;

    [ISystemManager]
    [RemoteClass(alias="dto.ConsultaUsrDTO")]
    [Bindable]
    public class UsrDTO
    {
        public function UsrDTO(){

        }

        public var idUsr:String;

        public var nombre:String;

        public var paterno:String;

        public var materno:String;

    }
}
```

Para llamar a nuestro objeto remoto desde la aplicación flex lo hacemos con el siguiente elemento:

```
<mx:RemoteObject id="ConsultaBO" destination="consultaUsr"
    showBusyCursor="true">
    <mx:method name="getUsuarios" fault="onFault(event)"/>
    <mx:method name="deleteUsr" result="eliminar_Result()"
        fault="onFault(event)"/>
</mx:RemoteObject>
```

Destination es el encargado de conectarse al f remoting-config.xml para llamar a sus objetos remotos java. y mediante la tag <mx:method> llamamos al metodo que esta definido en el .class.

De esta manera nuestro código para la pantalla de consulta queda así:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical" width="100%" height="100%"
    creationComplete="getServicio();">

    <mx:RemoteObject id="ConsultaBO" destination="consultaUsr"
        showBusyCursor="true">
        <mx:method name="getUsuarios" fault="onFault(event)"/>
        <mx:method name="deleteUsr" result="eliminar_Result()"
            fault="onFault(event)"/>
    </mx:RemoteObject>
```




Formato FOR-DES-401.-Flex Builder
V1.0.doc401
Flex Builder

Metodología: D.T.A.A.

Coordinador: Marco
García

Nuestro código de para el componente para agregar registro queda de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    width="400"
    height="200"
    showCloseButton="true"
    close="PopUpManager.removePopUp(this)"
    title="Nuevo Usuario" horizontalAlign="center"
    verticalAlign="middle" xmlns:local="*">

    <local:UsrDTO id="DTO"
        nombre="{nombreForm.text}"
        paterno="{paternoForm.text}"
        materno="{maternoForm.text}"
    />

    <mx:RemoteObject id="ConsultaBO" destination="consultaUsr"
        showBusyCursor="true">
        <mx:method name="saveUsr" result="guardar_Result()"
            fault="onFault(event)"/>
    </mx:RemoteObject>
```


Y nuestra clase para editar queda de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    width="400"
    height="250"
    showCloseButton="true"
    close="PopUpManager.removePopUp(this)"
    title="Nuevo Usuario" horizontalAlign="center"
    verticalAlign="middle" xmlns:local="*">

    <local:UsrDTO id="DTO"
        idUsr="{idForm.text.toString()}"
        nombre="{nombreForm.text}"
        paterno="{paternoForm.text}"
        materno="{maternoForm.text}"
    />

    <mx:RemoteObject id="ConsultaBO" destination="consultaUsr"
        showBusyCursor="true">
        <mx:method name="updateUsr" result="actualizar_Result()"
            fault="onFault(event)"/>
    </mx:RemoteObject>
```

Podemos observar que utilizamos este componente <local:UsrDTO> el cual apunta a nuestra clase UsrDTO.as la cual se enlaza a nuestros DTO mediante el ya conocido remoting-config.xml.

	Formato FOR-DES-401.-Flex Builder V1.0.doc401 Flex Builder	Metodología: D.T.A.A.
		Coordinador: Marco García



Recuerde que nuestros .class al igual que los paquetes se deben encontrar en la carpeta classes del WEB-INF para que puedan ser llamadas, puede verificar su programación e incluso descomprimir dentro de la carpeta de webapps del tomcat en el zip de nombre FlexRemote.rar dentro la carpeta de inicio de este tutorial.