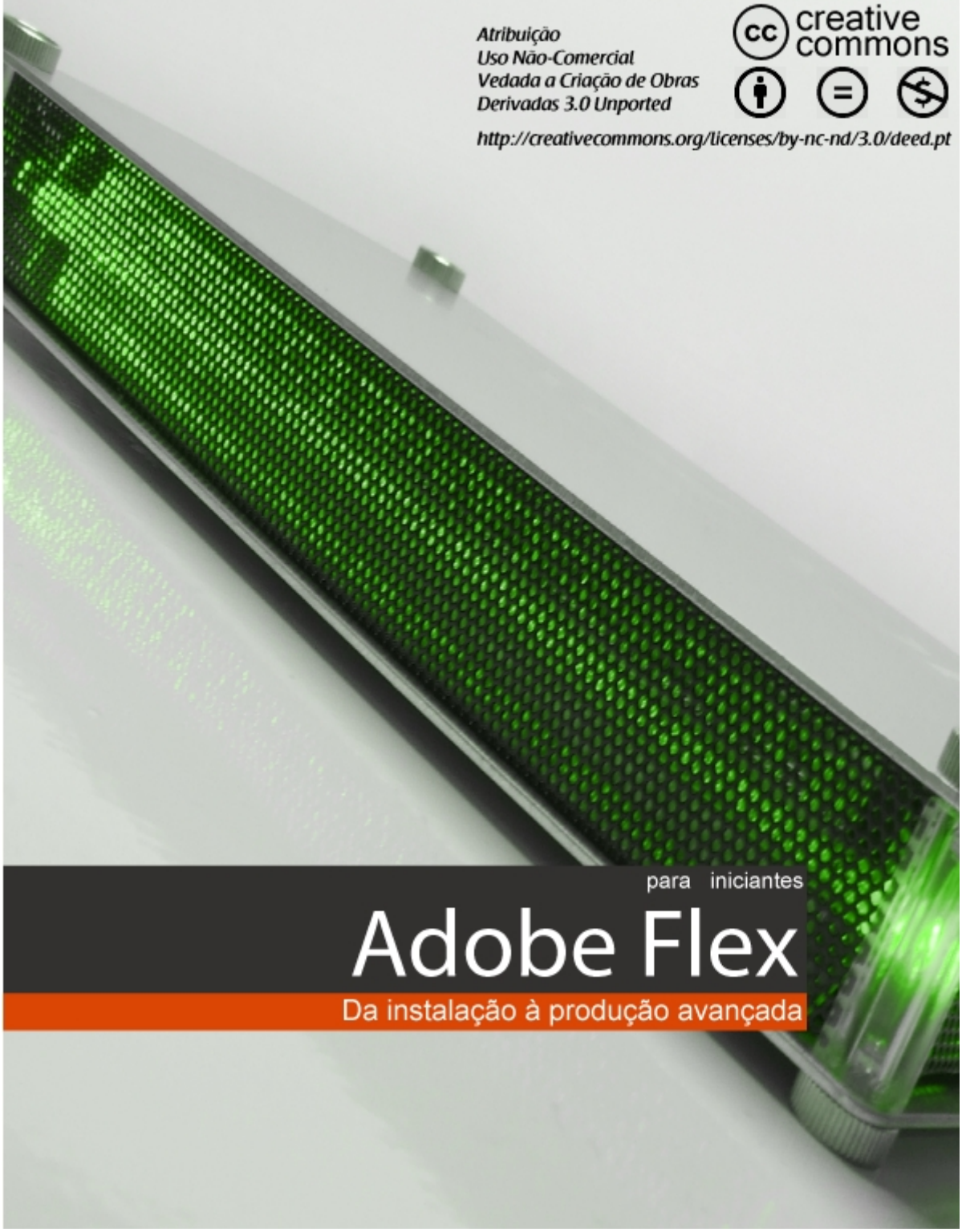
Mário Santos
www.msdevstudio.com



para iniciantes

Adobe Flex

Da instalação à produção avançada

Adobe Flex a Partir do Zero

Desde a instalação à produção avançada.
Elaborado por Mário Santos

INDEX

1. Observações iniciais.

Bom, como as informações relativas a este "framework" são demasiado escassas, principalmente para quem se está a iniciar na área das RIA's (Rich internet Applications ou aplicações ricas para internet), mesmo não tendo muito tempo livre nem sendo um profissional vou decidi partilhar um pouco do que já aprendi para quem deseja começar do zero a construir os seus projectos em Flex, focando essencialmente o aplicativo Adobe Flex Builder.

O que é uma RIA?? Passo a citar:

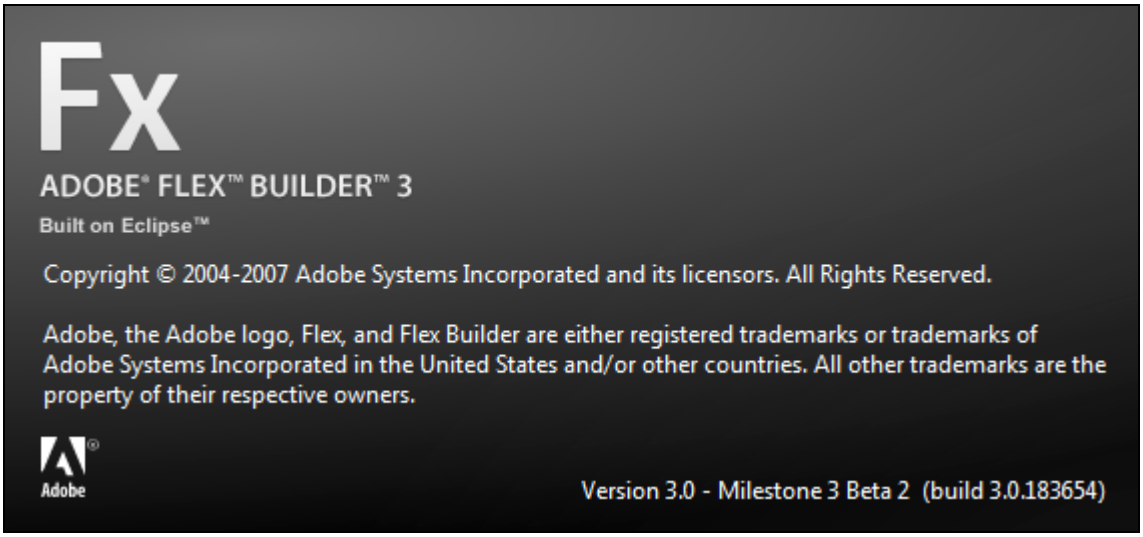
"RIA é a abreviação de Rich Internet Applications ou Aplicações Ricas para Internet.
É uma Aplicação Web que contêm características e funcionalidades de uma aplicação desktop tradicional.
Tipicamente uma aplicação RIA transfere a necessidade de processamento do cliente (numa arquitectura cliente-servidor) para o navegador mas, mantém o processamento mais pesado no servidor de aplicação.
O termo RIA foi usado pela primeira vez em 2001 pela Macromedia (hoje Adobe Systems).
Características:
-roda em um navegador e não requer a instalação de programas adicionais;
-roda localmente em um ambiente seguro chamado sandbox;"

Origem: Wikipédia, a enciclopédia livre.

Bom, o Adobe Flex é uma plataforma de desenvolvimento de RIA's tendo como principal base de distribuição das suas aplicações o Adobe Flash Player. A grande diferença do convencional flash e esta plataforma é mesmo o estilo de "programação" e a diversidade do conteúdo que é possível criar com o flex, já que cada vez mais são disponibilizadas API's para interacção com esta plataforma como o flicker, google maps, yahoo maps, youtube entre tantas outras.

O flex por si não é o mais indicado para construir animações frame-by-frame, para isso podem usar o flash e usar a vossa animação dentro do Flex, mas sim para a construção de RIA's.

Este documento será dividido em varias secções, abrangendo os mais variados assuntos como a instalação, animações/transições simples, programação "action script", comunicação com objectos remotos ou criação de uma aplicação para desktop via Adobe AIR.



Utilizarei o Flex Builder 3 beta 2 (Trial, Windows) que para iniciação é o mais indicado, já que o flex sdk (Gratuito) é muito mais "complexo" para iniciação visto a ausência da parte gráfica, e o Flex Builder 3 alpha (linux) ainda não tem parte de componentes drag & drop.

Neste manual irei usar imagens e código fonte sempre derivados da secção anterior do documento, por isso aconselho a leitura e execução de todos os passos pela sua devida ordem, caso contrario poderão não entender partes/exemplos de código porque estes foram criados anteriormente.

Este manual tem como objectivo ensinar ao utilizador todos os passos básicos para a iniciação na plataforma Flex, mas também no Flex Buider, se seguirem a leitura atentamente, e ao mesmo tempo efectuarem todos os passos indicados no final da leitura já conseguirão programar e ter uma facilidade enorme em compreender e trabalhar com o Adobe Flex.

2. Instalando, configurando e fazendo o típico teste "hello world".

Bom, se ainda não tiverem feito o download do Flex Builder beta 2, façam-no antes de prosseguir com a leitura, o download está disponível aqui:

http://labs.adobe.com/technologies/flex/flexbuilder3/

Para iniciar o download, clicam no link *"Get Flex Builder*" do lado direito do site, é necessário o registo (gratuito) para fazer o download (de cerca de 300mb de flex builder) Retirem a versão "stand alone" para windows. E instalem-o com os parâmetros "default". Não vou usar o flash debugger para já, mas se vos for proposto na instalação instalem-o também, pode ser-vos proposto para instalar os plug-in's para o Internet Explorer e Mozilla Firefox, por isso instalem-os também se usarem um destes "browsers".

Depois do Flex instalado, abram-no.. se na primeira janela (ao centro) aparecer uma mensagem parecida com "***Alternate HTML content should be placed here. This content requires the Adobe Flash Player. Get Flash*** " cliquem no link que vos é proposto (Get Flash) e façam o download/instalem o flash player desse site que se abrirá dentro do flex.

No final da instalação reiniciem o flex.

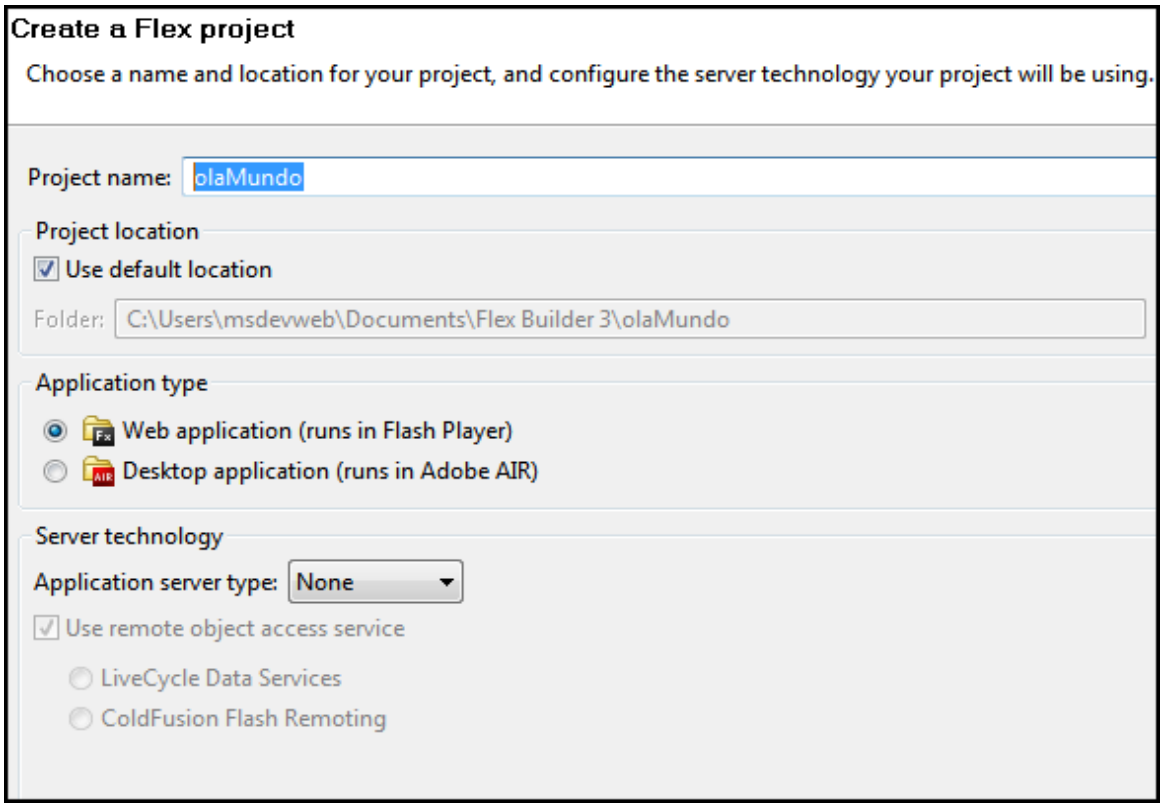
Caso apareça a mesma mensagem após reiniciarem o flex, vamos instalar o flash player de novo, mas desta vez a partir daqui: **http://labs.adobe.com/downloads/flashplayer9.html**

Dessa pagina instalem:
-**Active X Control para o Internet Explorer**
-**Plugin para Windows.**

Fechem o flex builder após a instalação do flash player, e reiniciem o flex builder, se tudo tiver corrido bem, já não deve aparecer o tal erro falado antes, mas sim uma página de bem-vindo.

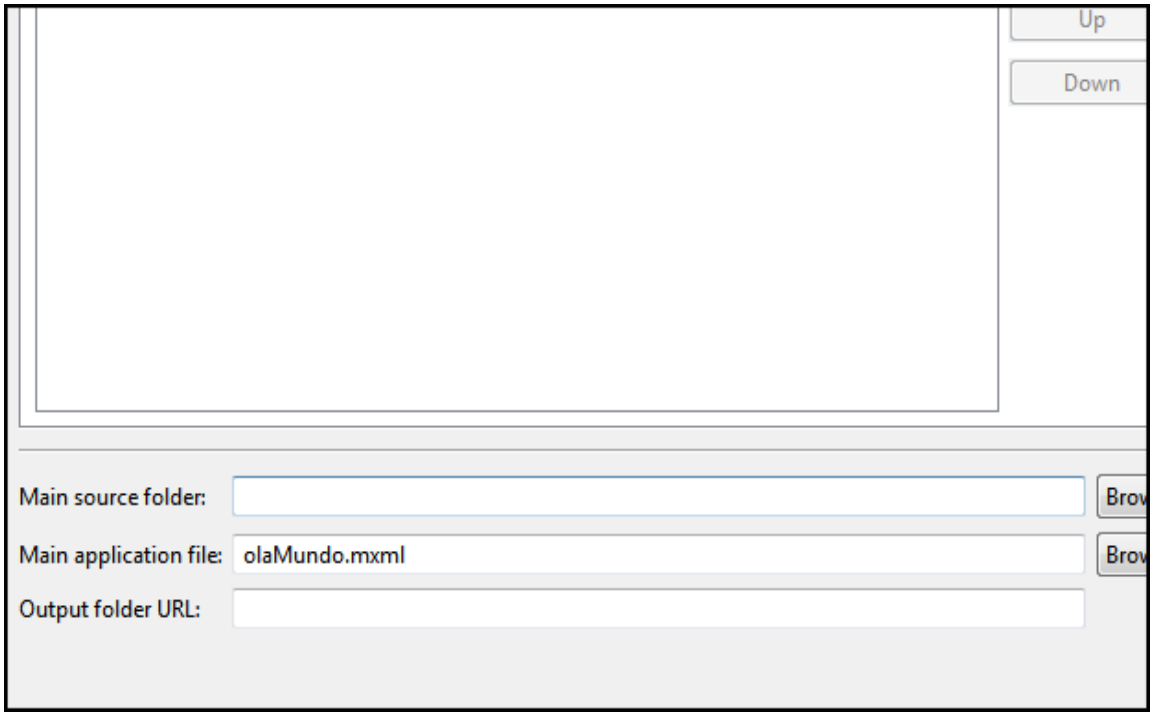
2.1. Criando um novo projecto.

Bom, começamos por criar um novo projecto, vão ao menu File -> New -> Flex Project. coloquem "olaMundo" no nome do projecto, como na figura em baixo :



Logo a seguir deixem o resto das informações como estão (para já). Cliquem em "Next".

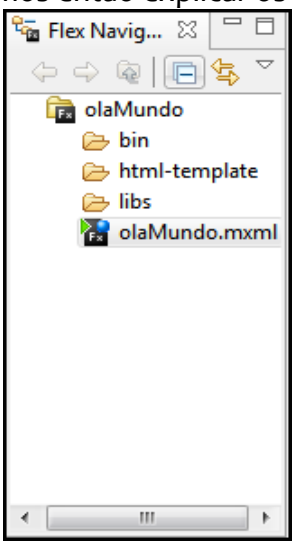
Na janela a seguir ao fundo onde está o source folder: "src" retirem e deixem o campo em branco :



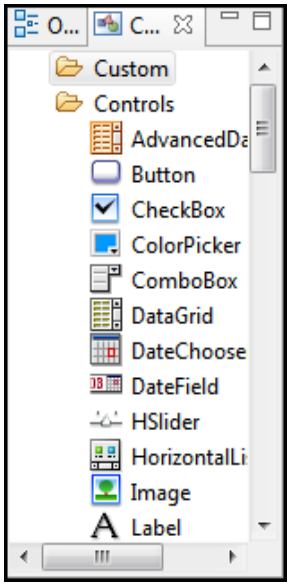
Cliquem em "Finish". O vosso projecto está agora criado...

2.2. Entendendo o espaço de trabalho do Flex Builder

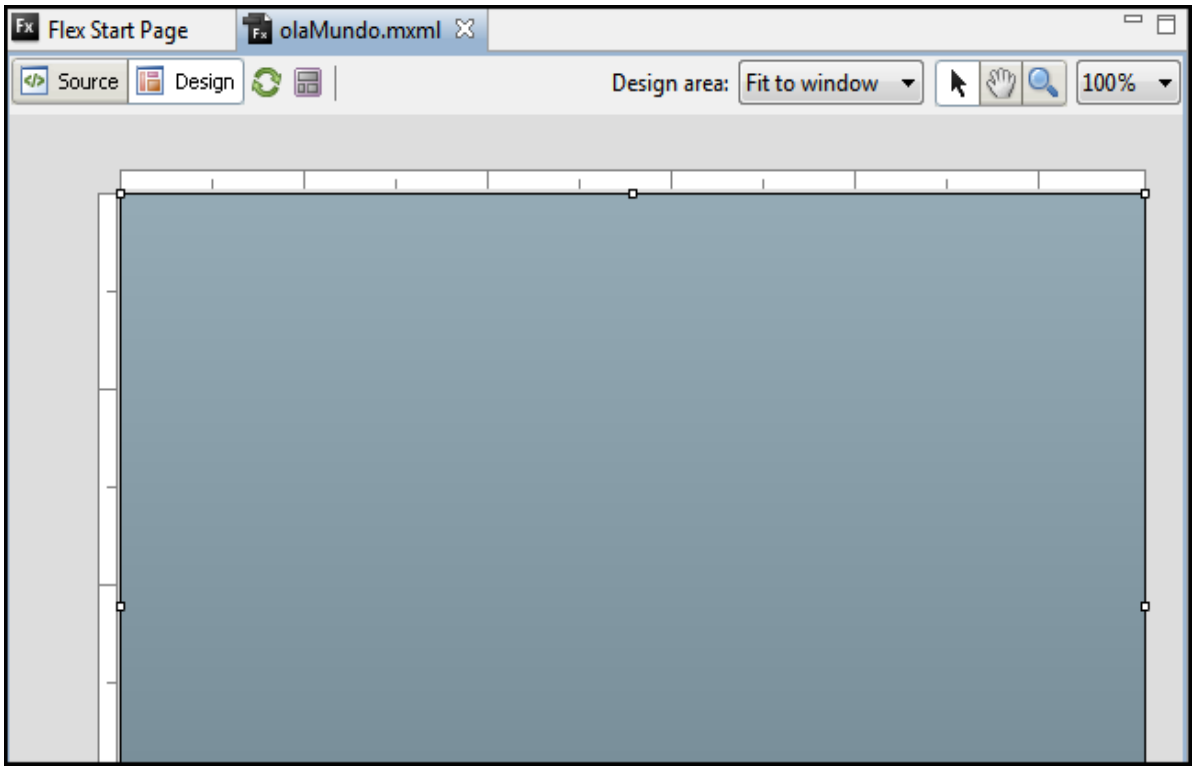
Vamos então explicar os painéis frontais do Flex Buider começando pelo lado esquerdo.



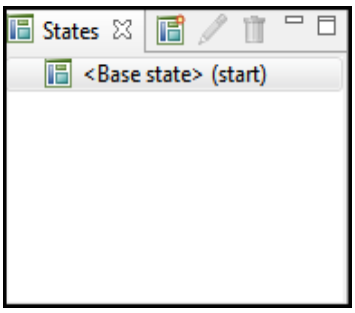
Flex Navigation, onde é listado os ficheiros e pastas que compõem o vosso projecto, se repararem foi criado um ficheiro olaMundo.mxml com um símbolo de play e um pequeno globo. Significa que esse documento é a aplicação principal do nosso projecto, é a que será sempre executada por defeito.



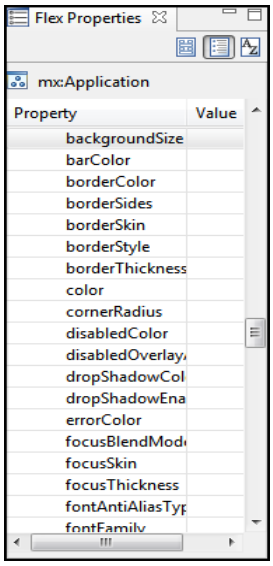
Components, onde pode encontrar todos os componentes gráficos para criação do "layout", onde podem encontrar de tudo um pouco desde botões, janelas, painéis, listas, grelhas, barras de navegação...etc... Todos estes componentes só podem ser usados "drag & drop", do tipo arraste e solte.



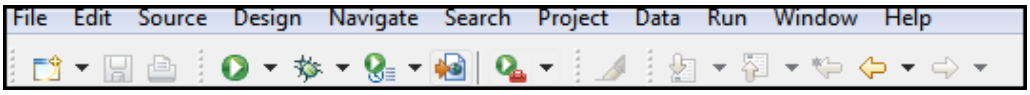
Ao centro está a área principal (zona de desenho) da vossa aplicação, onde existe também um botão de "Source" que ao ser clicado mostra o código fonte do layout, quem está familiarizado com o Adobe Dreamweaver não vai achar nada estranho e o botão "Design" mostra o layout da aplicação...
Cada vez que forem actualizados dados no layout eles são automaticamente adicionados no código, e vice-versa.



Do lado direito está presente a janela de "States", basicamente é uma janela onde são mostradas todas as "páginas"/"estados" da nossa aplicação, funciona como se fossem páginas como em html. pagina 1, pagina 2 e vai-nos permitir mais à frente fazer links entre eles.



Janela de configuração ("Properties"), onde podem definir cores, bordos, efeitos, margens, tipos de letra, alinhamento, tamanhos. etc... para acesso a cada uma das propriedade de cada elemento do nosso layout, temos que clicar em cima, por exemplo se clicarem no fundo "azul" da área de desenho, do lado direito têm acesso às propriedades desse mesmo fundo, que nesse caso é o corpo de base da nossa aplicação.



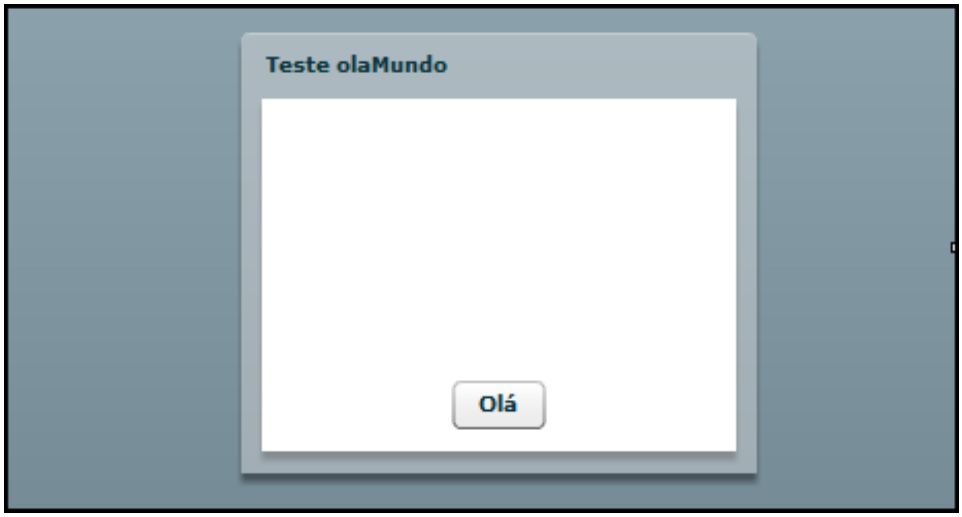
Bom, a descrição termina por mostrar a barra horizontal que se encontra no topo, essencialmente vamos focar os seguintes botões:

Da esquerda para a direita:
-O primeiro botão é o que permite compilar a nossa aplicação, se carregarem nele, ser-vos-à apresentada uma janela de compilação, onde não requererá atenção se não houver alterações na aplicação desde que foi compilada pela ultima vez... caso não tenham sido salvos dados, ser-vos-à proposto uma janela com os ficheiros não salvos, e indiquem para salvar todos, seleccionando-os (por defeito), podem aplicar a opção de salvar sempre antes de compilar.
No final da compilação (que demora um pouco) será aberta uma janela do "browser" Internet Explorer (por defeito) ou do vosso "browser" predefinido com o resultado da vossa aplicação.

-O Segundo botão é o de debug, que comunicará com o flash player e a sua componente debug se esta tiver sido instalada na altura da instalação do flex. Não vou falar muito deste porque não o vamos usar neste tutorial.
-O terceiro é o profile, onde o flex abre o flex profiler, serve basicamente para controlar os aspectos de desempenho do flex... também não vou falar para já.
O quarto é o Export release version, que falarei mais à frente quando falar do Adobe AIR.
O quinto e ultimo permite definições de configuração de ferramentas externas, para já não vou falar...

2.3. Criando o primeiro Exemplo "hello world".

Coloquem a vossa área de trabalho em modo "Design", clicando no botão "Design".
Da janela de componentes, procurem um componente "Panel" (em layout components), cliquem nele e arrastem-no para a vossa área de desenho; cliquem 2x com o botão esquerdo do rato no topo desse painel e escrevam "Teste olaMundo", de seguida da mesma janela de componentes procurem um "Button" e arrastem para cima do painel que criaram na área de desenho, cliquem 2x nesse mesmo botão e mudem o nome para "Olá", deverão obter um resultado como na figura em baixo.



A parte gráfica do nosso projecto "hello world" está feita, vamos passar a explicar a estrutura do código fonte do que já fizemos até agora.

2.3.1. Entendendo a estrutura MXML

No topo vamos clicar no botão "Source" e vai aparecer o código do nosso layout, estará algo muito parecido, variando apenas as posições “x=”, “y=”, como o seguinte:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
      <mx:Panel x="126" y="114" width="250" height="200" layout="absolute"
title="Teste olaMundo">
                <mx:Button x="92.5" y="128" label="Olá"/>
       </mx:Panel>
</mx:Application>
```

Como podem ver, temos <mx:Application ...> que é a nossa aplicação/fundo.
Aí dentro da "tag" temos um <mx:Panel ...> é o nosso painel, e dentro da "tag" Painel o nosso <mx:Button .. /> note que no botão temos no final da sua definição (x=posição eixo dos xx e y=posição eixo yy) temos uma "/" antes do ">" quer significar que como não vamos colocar nada dentro dele, fechamos logo a "tag" em vez de usar </mx:Button>.
As tags faladas contêm dentro delas as definições como x, y, width, height, e podemos adicionar mais definições, mas agora isso não tem importância... falamos mais à frente...
No final, temos que fechar as "tag's" mx pela sua devida ordem, como já fechamos a "tag" button (por defeito "/"), resta fechar o painel e a aplicação:

```
        </mx:Panel>

</mx:Application>
```

Bom, já compreendemos um pouco da estrutura em cadeia, estilo XML, mas no flex chamado de MXML.
Se quisermos colocar outro botão no nosso painel, mas agora via MXML, basta escrever dentro da tag <mx:Painel> o seguinte:

```
<mx:Button label="botão de teste" x="50" y="128" />
```

Se repararmos, ao escrever "<mx:" aparece uma lista de "componentes", a mesma lista da janela de onde arrastamos o nosso painel e o nosso botão, basta seleccionar o "Button". Nessa lista podemos adicionar qualquer um dos outros componentes gráficos, mas para já ficamos pelo Button. O nosso código ficaria assim:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
   <mx:Panel x="126" y="114" width="250" height="200" layout="absolute"
title="Teste olaMundo">
      <mx:Button x="92.5" y="128" label="Olá"/>
               <mx:Button label="botão de teste" x="50" y="128" />
   </mx:Panel>
</mx:Application>
```

Se clicarem agora no botão de "Design" ao cimo, aparece o nosso layout, já com o nosso novo botão. De volta ao código (botão "Source") apaguem o botão que acabaram de criar.

2.3.2. Criando um script (AS3)

Vamos agora criar um script muito simples.

Logo em baixo após a tag </mx:Panel> e antes da </mx:Application> escrevemos o seguinte: <mx: e aparecerá uma lista que já foi falada em cima, seleccione o "Script" dessa mesma lista e deverá ser adicionado automaticamente o seguinte:

```
<mx:Script>
<![CDATA[

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

Ai dentro podemos definir comentários ao código para nos ajudar a compreender melhor, este tipo de comentários já é bem conhecido no mundo da programação, aqui usamos "//" para definirmos os nossos comentários:

```
<mx:Script>
<![CDATA[

      //nosso comentário aqui

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

O nosso objectivo é que ao clicar no botão "Olá" que criamos anteriormente, seja apresentada uma mensagem que diga "Mundo!", para isso usamos um componente do flex chamado Alert. Antes de usarmos qualquer "componente/package" do flex em Action Script temos que importar o devido componente do devido "pacote"/"package" para o projecto, por isso escrevam dentro desse script :

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

Se repararem ao escrever "mx. " vao aparecer todos os controlos, funções, eventos, etc... do flex/flash divididos por categorias ("packages"), se escrever "mx.controls." mostra na lista todos os objectos possíveis de importação do "package" controls do "mx." bom, no final dentro desse script fica algo como:

```
<mx:Script>
   <![CDATA[

      import mx.controls.Alert;

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

A seguir ao nosso import vamos criar uma função para ser chamada quando o nosso botão for clicado..

Escrevam logo após ao import:

```
private function aoClickar():void {
            Alert.show("Mundo");
            }
```

Explicando, é uma função "privada" de uso na aplicação, com o nome "aoClickar" que não vai retornar nada (":void").

Dentro dessa função escrevemos o que queremos que seja feito quando a função for executada, neste caso apresentar um alerta, logo chamamos o nosso "objecto" Alert que foi importado em cima, e dizemos-lhe que o queremos mostrar:

```
Alert.show("Mundo");
```

Dentro dos parâmetros da função, que vos são apresentados, apenas queremos utilizar uma mensagem para já, sem definir botões nem títulos nesse "alert."

NOTA IMPORTANTE:
Todos os imports/componentes em modo de edição são "case sensitive", ou seja alert.show("Mundo"); não funciona, tem que ser Alert.show("Mundo");

No final de tudo deve estar algo como:

```
<mx:Script>
   <![CDATA[

      import mx.controls.Alert;

      private function aoClickar():void {
            Alert.show("Mundo");
            }

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

Temos quase o nosso teste completo, só falta "dizer" ao botão "Olá" que ao ser clicado tem que chamar esta nossa função. Para isso vamos indicar na "tag" <mx:Button ..> o seguinte:

```
click="aoClickar();"
```

Ficará algo como:

```
<mx:Button x="92.5" y="128" label="Olá" click="aoClickar();"/>
```

Ora, vamos salvar, Menu File->Save

Uma verificação final ao nosso código que deve estar como em baixo, tirando os parâmetros x, y, width e height.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
   <mx:Panel x="126" y="114" width="250" height="200" layout="absolute"
title="Teste olaMundo">
      <mx:Button x="92.5" y="128" label="Olá" click="aoClickar();"/>
   </mx:Panel>

   <mx:Script>
      <![CDATA[

         import mx.controls.Alert;

         private function aoClickar():void {
         Alert.show("Mundo");
         }

      ]]>
   </mx:Script>

</mx:Application>
```

No "Design" não alterou nada.

Clicam no primeiro botão daqueles 5 no topo do Flex Builder que falei em cima |  |, esperem a compilação, o "browser" vai arrancar e apresentar o nosso projecto já compilado e a correr em Flash.

A Sua primeira aplicação FLEX está concluida, experimente clicar no Botão "Olá".

3. Entendendo a ordenação e estruturação do código.

Bom, quero antes de mais lembrar que vamos usar por enquanto o exemplo criado anteriormente. Tínhamos criado a nossa primeira aplicação... bom, neste exemplo vou aprofundar mais um pouco da estrutura e organização de código... já antes falei um pouco, mas agora vou exemplificar mais um pouco...
O que aconselho, é estruturar e organizar bem o código por uma questão de fácil identificação de erros, e também por uma questão de desempenho.
Exemplo de mau código de um script:

```
<mx:Script>
<![CDATA[
 private function clickado():void {
 var x:String;
 x="Olá";
import mx.controls.Alert;
Alert.show(x+"Mundo");

 }
  ]]>
</mx:Script>
```

Bom, se alguns de vocês estão habituados a boas práticas de programação, já repararam que podia ter simplificado bastante esta função começando pela tabulação adequada do código (estilo tree). Não vou estar a aprofundar muito este ponto, porque tornar-se-ia muito extenso e até porque o próprio flex builder informa muitos destes erros. Ora reparem:

```
<mx:Script>
<![CDATA[
	import mx.controls.Alert;

	private function clickado():void
	{
	 var x:String = "Olá Mundo";
	  Alert.show(x);
	}

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

Como podem ver, as coisas simplificaram-se muito... principalmente a nível de erros (não se deve usar o import dentro de uma função) porque seria importado cada vez que a função for chamada sobrecarregando o aplicativo e invalidando o código.
Depois, ao declarar a string, podemos logo atribuir-lhe um valor, e neste caso como queríamos apresentar um alerta com a frase "Olá Mundo" declaramos logo a variável com esse valor, e dentro do alert chamamos a variável... podia ainda simplificar mais, usar somente o Alert.show("Olá Mundo"); mas não o fiz para vocês aprenderem já a declarar variáveis.
Como devem ter reparado (se não fizeram copy/paste) ao escrever "var x:" apareceu uma lista com todos os tipos de variáveis que podem ser criadas/declaradas. O código foi organizado também em forma de "tree", devidamente tabulado e estruturado.
Uma das grandes diferenças da maior parte das linguagens é que podemos atribuir objectos a uma variavél, por exemplo:
var x:Panel;

só isto não funcionaria devidamente porque esse componente não existe no script, nem foi devidamente atribuído. Este tipo de declarações de variáveis como objectos é muito útil, já que podemos inserir objectos no nosso layout por via de Action Script (já devem estar a imaginar centenas de formas úteis e essenciais que tornam isto muito importante), mas falarei disto mais à frente. Bom, falámos da má organização de scripts, agora vamos falar um pouco da má organização de MXML.
Imaginem no nosso antigo código, que criamos anteriormente:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
	<mx:Panel x="126" y="114" width="250" height="200" layout="absolute" title="Teste
olaMundo">
		<mx:Button x="92.5" y="128" label="Olá" click="aoClickar();"/>
	</mx:Panel>
	<mx:Script>
		<![CDATA[

			import mx.controls.Alert;

			private function aoClickar():void {
			Alert.show("Mundo");
			}
		]]>
	</mx:Script>
</mx:Application>
```

Imaginem que queremos outra função? não devemos criar de novo outro <mx:Script>, mas sim usar o existente para a criação de outra função, mesmo assim existem alguns pequenos pormenores que muitas vezes falham.
Se não usarmos a organização como em cima, torna-se mais complicado, por exemplo, identificar devido painel, ou botão.

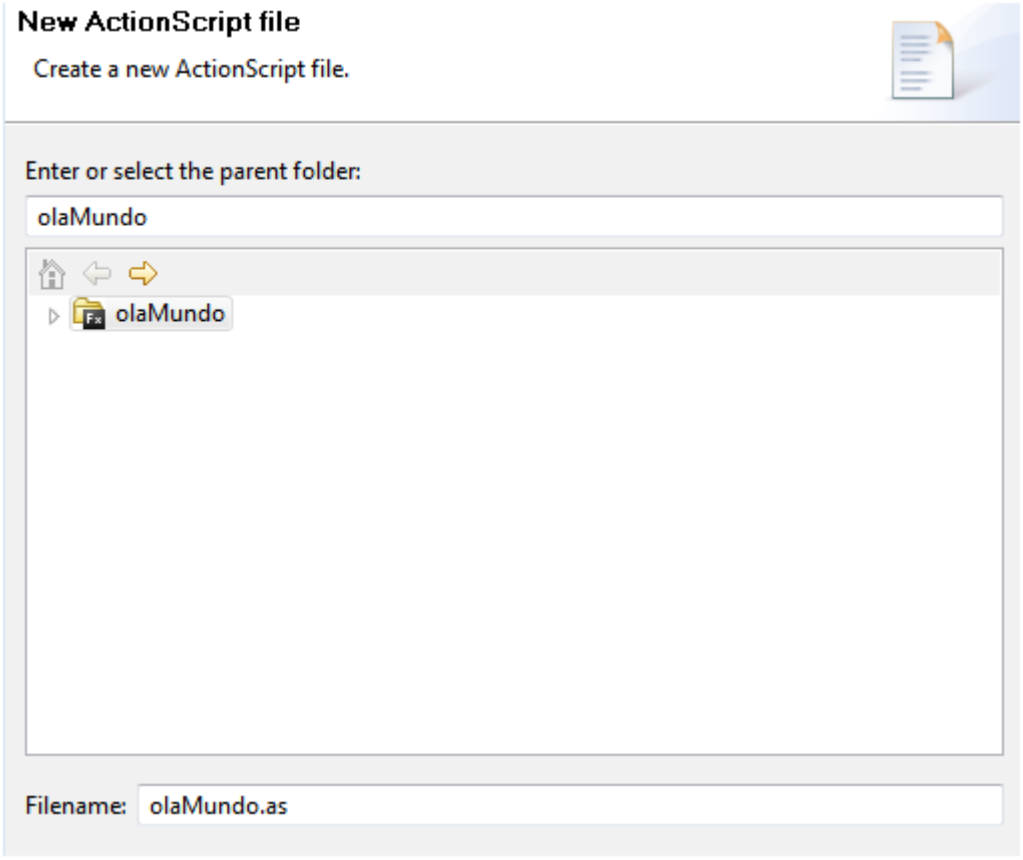
Bom, a nível de organização de MXML pouco mais há a dizer a não ser usar as boas práticas e "tabular" devidamente o código, para fácil uso, e não usar por exemplo a importação de componentes que não sejam necessários na aplicação, por exemplo, o uso do import mx.controls.Alert; e não depois não o usarmos no nosso código. (Isto é uma falha muito comum, e que torna a compilação muito mais demorada e mais sujeita a erros).

Podemos simplificar mais ainda a criação dos nosso aplicativo a nível de código, criando para isso um documento à parte para todo o conteúdo Action Script que vou passar a explicar. (neste caso não é necessário o uso de um ficheiro externo contendo o nosso action script, visto que se trata de de um script muito simples, mas para scripts extensos é a melhor opção).

3.1. Criação de um Ficheiro Action Script Externo para uso no Flex.

Vão ao menu "File" -> "New" -> "ActionScript File"

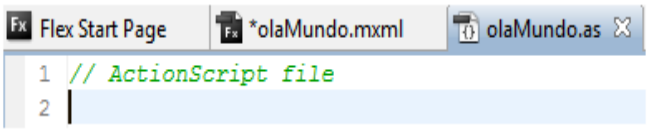
Na janela que se irá abrir, coloquem em FileName (ao fundo) "olaMundo" como na figura em baixo:



Cliquem em "Finish".

A nossa área de desenho abriu agora o nosso recente criado "olaMundo", contendo apenas
//ActionScript File
e do nosso lado esquerdo, no "Flex Navigator" foi criado o nosso olaMundo.as (a extensão indica que é um ficheiro ActionScript).
Como este exemplo serve para mostrar como criar o nosso script num ficheiro à parte e usa-lo no nosso aplicativo, vamos limitar-nos a copiar o nosso script para este recente criado olaMundo.as

Se repararem, no topo da área de desenho estão abertos os nosso ficheiros do projecto que estamos a trabalhar como na figura ao lado.



Vamos clicar no olaMundo.mxml, e cortar o nosso script do "Source" e cola-lo no olaMundo.as, ficaríamos com o seguinte código no olaMundo.mxml:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
	<mx:Panel x="126" y="114" width="250" height="200" layout="absolute"
title="Teste olaMundo">
		<mx:Button x="92.5" y="128" label="Olá" click="aoClickar();"/>
	</mx:Panel>
	<mx:Script>
		<![CDATA[

		]]>
	</mx:Script>
</mx:Application>
```

e no nosso olaMundo.as :

```
import mx.controls.Alert;

private function aoClickar():void
{
	Alert.show("Mundo");

}
```

Agora o que temos a fazer, é chamar o nosso script externo olaMundo.as no nosso olaMundo.mxml, para isso voltamos ao "Source" do olaMundo.mxml e na "tag" <mx:Script> vamos remover todo o seu conteúdo e remover também a "tag" </mx:Script> e colocar o seguinte:

```
<mx:Script source="olaMundo.as" />
```

notem que indicamos na "tag" o parâmetro "source" a indicar onde se encontra o código fonte desse script, e no final colocamos o caracter "/" porque não utilizaremos qualquer script na sua área.
Salvem os documentos. (olaMundo.mxml e olaMundo.as) no menu "File" -> "Save All" ou com as teclas de atalho CRTL+SHIFT+S.

Após salvarem os vossos ficheiros Clicam no botão para compilar a vossa aplicação, se tudo correu bem devem ter o mesmo exemplo a correr, mas agora usando o script importado.
Isto é muito importante, já que no desenvolver de uma aplicação de tamanho médio/grande podemos separar o nosso código action script para mais fácil identificação e para futuramente podermos usar também esses scripts em outros projectos.

4. Entendendo os componentes internos, states e transições/efeitos.

Uma das partes mais difíceis da arquitectura do flex é a sua organização interna de "Packages"/"Componetes", como já disse e exemplifiquei anteriormente, para usarmos o Alert, temos ante de o importar para a nossa aplicação para ele estar disponível para uso, caso contrario ao compilar seriam apresentados erros.

Bom, todos os componentes disponíveis mo modo gráfico, na janela componentes, estão também disponíveis em MXML e em Action Script, mas aqui já com algumas alteração para que sejam apresentados no nosso layout.
Vamos ver um exemplo para adicionar um novo painel na área de desenho pelo modo de MXML e pelo modo de Action Script. Usando o nosso código:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute">
	<mx:Panel x="126" y="114" width="250" height="200" layout="absolute"
title="Teste olaMundo">
		<mx:Button x="92.5" y="128" label="Olá" click="aoClickar();"/>
	</mx:Panel>
	<mx:Script source="olaMundo.as" />
</mx:Application>
```

Vamos alterar o nosso tamanho da nossa aplicação para se adaptar ao tamanho do browser para que possamos ver melhor a nossa área, na "tag" <mx:Application ..> vamos escrever/alterar os seguintes parâmetros width e height para 100%, depois de alterado ficaria algo como:

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="100%" height="100%">
```

de seguida vamos escrever o seguinte a seguir à nossa "tag" <mx:Script> a antes da <mx:Application> o seguinte:

```
<mx:Panel x="400" y="114" width="180" height="115" layout="absolute"
title="Painel adicionado via MXML" />
```

Reparem que coloquei a "/" no final, o que quer dizer que o nosso painel para já não terá qualquer conteúdo, de seguida vejam o modo "Design" e verifiquem que o nosso painel já foi adicionado, ficando algo como a figura em baixo:

Agora vamos adicionar um terceiro painel, mas desta vez via action script, e só irá aparecer no nosso layout quando clicamos no botão olá.
No nosso olaMundo.as vamos aproveitar o nosso já criado olaMundo.as e alem de mostrar um novo painel vamos já adicionar um pequeno efeito (WipeDown) na altura da criação desse painel, um efeito disponível como package no flex.
Escrevam o seguinte no olaMundo.as a seguir ao nosso import do Alert:

```
import mx.containers.Panel; //vamos importar o componente painel
import mx.effects.Effect;   //importar a definição effect
import mx.effects.WipeDown; //importar o efeito
```

à seguir à função aoClickar(); criamos uma nova função addPanel(); :

```
private function addPanel():void
{
	var novo:Panel = new Panel; //declaramos a variavel novo como painel
	var efeito:Effect = new WipeDown; //decl. Efeito com novo wipeDown effect
	novo.width=180;  //comprimento
	novo.height=115; //altura
	novo.x=0; //posição x onde o painel vai aparecer
	novo.y=0; //posição y onde o painel vai aparecer
	novo.id="panel3"; //identificamos o painel como "panel3"
	efeito.target=novo; //definimos o destino/target do efeito wipedown
	this.addChild(novo); //adicionamos como "filho" do "stage"/Mostra painel
	efeito.play(); //e finalmente iniciamos o efeito wipeDown


}
```

Se lerem atentamente, e ao escreverem, verão muitas das funções disponíveis, bem como componentes e efeitos, mas restringam-se apenas à função em cima.
Lembrem-se que temos sempre que importar os componentes/efeitos/instâncias sempre que as pretendemos usar, mas apenas uma vez. O nosso olaMundo.as deve estar algo como:

```
// ActionScript file
import mx.controls.Alert;
import mx.containers.Panel;
import mx.effects.Effect;
import mx.effects.WipeDown;

private function aoClickar():void
	{

	Alert.show("Mundo");

	}

private function addPanel():void
{
	var novo:Panel = new Panel;
	var efeito:Effect = new WipeDown;
	novo.width=180;
	novo.height=115;
	novo.x=0;
	novo.y=0;
	novo.id="panel3";
	efeito.target=novo;
	this.addChild(novo);
	efeito.play();

}
```

Agora no nosso olaMundo.mxml vamos criar um novo botão via MXML para chamar a nossa função para criar o novo panel, escrevam dentro do nosso primeiro painel, em baixo da "tag" do nosso botão "Olá" o seguinte:

```
<mx:Button x="80" y="148" label="Adiciona Painel" click="addPanel();"/>
```

O vosso código deve estar como o seguinte:

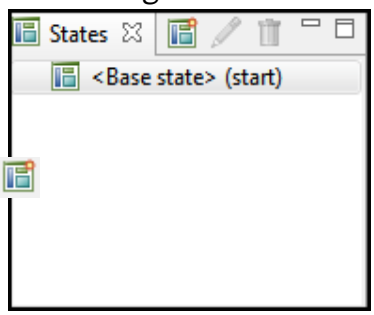
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="609" height="393">
	<mx:Panel x="107" y="89" width="250" height="200" layout="absolute"
title="Teste olaMundo">
		<mx:Button x="92.5" y="128" label="Olá" click="aoClickar();"/>
		<mx:Button x="80" y="148" label="Adiciona Painel"
click="addPanel();"/>
	</mx:Panel>
	<mx:Script source="olaMundo.as" />
	<mx:Panel x="365" y="89" width="180" height="115" layout="absolute"
title="Painel adicionado via MXML" />
</mx:Application>
```

Devemos ter agora um novo botão "Adiciona Painel" no nosso layout que ao ser clickado chama a função addPanel(); que criamos anteriormente. Salvem os vossos ficheiros (CRTL+SHIF+S) e corram a aplicação ; Devem estar disponíveis apenas 2 painéis, o nosso painel "Olá Mundo" e o "Painel criado via MXML". Se clicarem no botão "Adiciona Painel" verão a criação de um novo painel, via Action Script com o efeito WipeDown do flex. Cada vez que clicarem será adicionado um novo painel, na mesma posição, com o mesmo efeito.

4.1. States, Entendendo a sua disposição

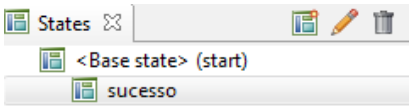
Já vimos como adicionar um painel ao nosso "stage" principal por via de action script chamado através de um botão, vamos agora iniciar uma pequena demonstração de "states". Os states, são os "estados" da nossa aplicação, funcionam como sub-paginas em html as quais chamamos assim que formos precisando delas.

Vamos focar a janela "States" que foi apresentada em cima e mostrada na imagem em baixo. Nesta janela apenas se encontra no nosso "Base state" ou seja, a base da nossa aplicação e como podem verificar encontra-se com a indicação (start) que indica que este deve ser o painel a apresentar quando iniciar a nossa aplicação.



Vamos criar um novo state, para isso clicamos com o rato no ícone que se encontra nesse painel, é-nos pedido o nome desse novo state, uma opção based on que permite escolher a base desse painel, neste momento vamos deixar estar por defeito (base state) e também tem a opção "Set as start state" que ao clicarmos vamos definir esse painel com o principal, mas não vamos seleccionar essa opção por agora.

Vamos dar o nome de "sucesso", e manter as opções sem alterar nada e clicar apenas no botão ok. Devemos então ter na janela "States" algo como a imagem apresentada ao lado:



Se carregarem no base state e de novo no sucesso não acontece nada, porque os estados são iguais (o sucesso foi criado como base do base state). Seleccionamos o state sucesso e clicamos por exemplo no nosso painel com o titulo **Painel adicionado via MXML** e clicamos na tecla DEL, para apagar esse painel, agora se clicarem no "Base state" verão que o painel ainda se mantém lá, mas ao clicar no sucesso esse painel desapareceu. Ou seja, temos criados 2 estados da nossa aplicação. Se executarmos a nossa aplicação agora, veremos que se mantem tudo na mesma, porque o nosso painel que arranca em primeiro é o Base state e não o sucesso. Para arrancarmos o sucesso em primeiro teríamos que o definir como "set as start state", opção que podemos definir clicando no pequeno lápis no painel de states. Se verificarmos o nosso MXML (código) teremos algo como:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute" width="609"
height="393">
	<mx:states>
		<mx:State name="sucesso">
			<mx:RemoveChild target="{panel1}"/>
		</mx:State>
	</mx:states>
	<mx:Panel x="107" y="89" width="250" height="200" layout="absolute" title="Teste
olaMundo">
		<mx:Button x="39.5" y="108" label="Olá" click="aoClickar();"/>
		<mx:Button x="92.5" y="108" label="Adiciona Painel" click="addPanel();"/>
	</mx:Panel>
	<mx:Script source="olaMundo.as" />
	<mx:Panel x="365" y="89" width="180" height="115" layout="absolute" title="Painel
adicionado via MXML"  id="panel1"/>
</mx:Application>
```

se repararem foi adicionado automaticamente o código:

```
<mx:states>
      <mx:State name="sucesso">
            <mx:RemoveChild target="{panel1}"/>
      </mx:State>
</mx:states>
```

que indica que ao entrar nesse state (sucesso), o panel1 (Painel adicionado via MXML) será retirado (RemoveChild).
Vamos então agora adicionar um botão para alterar entre states, reaproveitando o mesmo código para alterar entre os 2 states. O reaproveitamento de código é muito importante em todas as linguagens de programação, porque além de reduzir o tempo de execução reduz também a "confusão" e extensão do nosso código.
Vamos colocar um botão no nosso painel (vou faze-lo via MXML), para isso no dento do nosso primeiro <mx:Panel> que aparece no código, acrescentamos o seguinte:

```
<mx:Button x="92" y="135" label="Muda estado" width="113.5"
click="mudaEstado()"/>
```

De seguida no nosso olaMudo.as vamos criar a nossa função, como a função vai ser usada para alternar entre os 2 paineis, teremos que verificar qual deles está "activo" para alternar para o outro, vamos então criar a nossa função:

```
private function mudaEstado():void
{
if(currentState!="sucesso") currentState="sucesso";
else currentState="";
//usamos o currentState="" para mudar para o state definido como (start)
}
```

poderíamos usar código action script directamente no evento click do nosso botão, como por exemplo:
click="currentState="sucesso";" (os " significam **""**), mas com este código não poderíamos utilizar o mesmo botão para mudar de estado, apenas seria utilizado para alterar para o estado "sucesso".
Se correrem agora a vossa aplicação verão que ao carregar no botão Muda Estado, o estado foi alterado para o sucesso (Que não apresenta o painel adicional via MXML), e ao clicar de novo o estado volta ao estado inicial (start).

Uma nota bastante importante, se desejarem alterar alguma coisa em determinado State devem selecciona-lo primeiro, e quando procederem a alterações devem verificar sempre se as alterações foram feitas no state correcto.

4.2. Transições e seus efeitos

Vamos agora falar um pouco de transições, o flex contem alguns efeitos possíveis de fazer nas transições/alternâncias entre os states.
Esses efeitos são executados quando se muda de state. Neste caso vamos apenas demonstrar 2 usos de efeitos um em cadeia (sequence) e outro em paralelo (parallel).

A primeira coisa temos que confirmar é a existência de uma identificação (id) nos nossos elementos que vão ser algo destes efeitos de transição. Neste caso vamos usar o primeiro painel **Teste Ola Mundo** e vamos acrescentar o id="panel2" assim:

```
<mx:Panel x="107" y="89" width="250" height="200" layout="absolute" title="Teste
olaMundo" id="panel2">
```

e no **botão olá** dar a identificação "botaoOla" assim:

```
<mx:Button x="39.5" y="108" label="Olá" click="aoClickar();" id="botaoOla"/>
```

E de seguida no final logo antes do </mx:Application> vamos escrever:

```
<mx:transitions>
            <mx:Transition fromState="*" toState="sucesso" id="TransicaoMudar" >
                  <mx:Sequence duration="1000" >
                        <mx:Glow target="{panel2}" alphaFrom="0" alphaTo="1"
color="#99cc00" />
                        <mx:Move yBy="25" target="{botaoOla}" />
                        <mx:Glow target="{panel2}" alphaFrom="1" alphaTo="0"
color="#99cc00" />
                        <mx:Move yBy="-25" target="{botaoOla}" />
                  </mx:Sequence>
            </mx:Transition>
      </mx:transitions>
```

Vamos começar por explicar, como este código que coloquei em cima serva apenas para demonstração, usei alguns efeitos e depois repus esses efeitos como de origem.

Iniciamos com <mx:transitions> de seguida podemos colocar mais que uma transição de 2 ou mais estados da nossa aplicação, depois temos a transição propriamente dita, que nos indica com o parâmetro **fromState** e **toState** quando deve ser executados os nossos efeitos, neste caso fromState="*" significa de qualquer state, qualquer alteração de states, com destino ao estado sucesso (toState="sucesso"), e de seguida colocamos as acções a efectuar nessa alteração. Aqui podemos utilizar principalmente 2 tipos de efeitos, efeitos sequenciais, um é executado a seguir ao outro, ou efeitos paralelos que são executados todos ao mesmo tempo.

Como indico no código:

```
<mx:Sequence duration="1000" >
                        <mx:Glow target="{panel2}" alphaFrom="0" alphaTo="1"
color="#99cc00" />
                        <mx:Move yBy="25" target="{botaoOla}" />
                        <mx:Glow target="{panel2}" alphaFrom="1" alphaTo="0"
color="#99cc00" />
                        <mx:Move yBy="-25" target="{botaoOla}" />
                  </mx:Sequence>
```

consegue-se perceber que temos uma sequência de efeitos, com a duração de 1 segundo (1000 ms), poderíamos definir aqui os objectos destinatários dos efeitos, mas como quero 2 efeitos diferentes, coloquei o destinatário (**target**) em cada um. Será efectuado primeiro um **mx:Glow** no **painel2**, de seguida um **mx:Move** para mover o **botaoOla** verticalmente 25px, logo depois um **mx:Glow** de novo para restaurar a cor original do painel e finalmente para restaurar a posição original do botão um **mx:Move**.

Todos estes efeitos são efectuados por ordem e nenhum é iniciado antes do anterior ter terminado. Vamos agora ver como efectuar a mesma animação, mas ao mesmo tempo e com os mesmos efeitos, para isso misturando efeitos paralelos.
Vamos escrever o nosso código assim:

```
<mx:transitions>
            <mx:Transition fromState="*" toState="sucesso" id="TransicaoMudar" >
                  <mx:Sequence duration="1000" >
                        <mx:Parallel duration="1000" >
                              <mx:Glow target="{panel2}" alphaFrom="0"
alphaTo="1" color="#99cc00" />
                              <mx:Move yBy="25" target="{botaoOla}" />
                        </mx:Parallel>
                        <mx:Parallel duration="1000" >
                              <mx:Glow target="{panel2}" alphaFrom="1"
alphaTo="0" color="#99cc00" />
                              <mx:Move yBy="-25" target="{botaoOla}" />
                        </mx:Parallel>
                  </mx:Sequence>
            </mx:Transition>
      </mx:transitions>
```

A ordem será:
Na transição de qualquer estado (fromState=”*”) para o estado sucesso (toState=”sucesso”) serão executada uma sequência (mx:Squence) de efeitos que engloba em primeiro a executado ao mesmo tempo (mx:Parallel) de 2 efeitos (mx:Glow e mx:Move) e de seguida será executada outra série de efeitos em paralelo para restaurar os efeitos como origem. Os efeitos estão agora agrupados em 2 mx:Parallel que serão executado por ordem (mx:Sequence), e de notar que os efeitos dentro do mx:Parallel são executados ao mesmo tempo e não um a seguir ao outro, como acontecia com o mx:Sequence.

5.Programação do exemplo olá mundo em Action Script

Iremos agora aprofundar um pouco o conhecimento de action script, bem como a disposição de childs no mais stage e iniciação aos eventListners.
Vamos criar o painel criado anteriormente (**teste OlaMundo**) via MXML, mas desta vez via Action Script, vamos apenas criar esse painel e não o **state Sucesso** e o **Painel adicionado via MXML.** Vamos então começar um novo action Script File (File-> New -> Action Script File) e vamos dar o nome de **mainScript** e dentro desse script vamos começar por criar uma função parar começar colocar os nossos elementos no Stage.

No **mainScript.as** escrevam: *// ActionScript file*

```
import flash.events.MouseEvent;
import mx.containers.Panel;
import mx.controls.Alert;
import mx.controls.Button;
import mx.effects.WipeDown;
import mx.effects.Effect;

private function criaTudo():void {
	//#########################################################
	//declaramos o painel numa nova variavel
	var painel2:Panel = new Panel();
	//definimos os parametros "graficos" do painel
	painel2.x=107;
	painel2.y=89;
	painel2.width=250;
	painel2.height=200;
	painel2.layout="absolute";
	painel2.title="Teste olaMundo";
	painel2.id="panel2";
	//adicionamos como filho do "mainStage"
	this.addChild(painel2);

	//#########################################################
	//vamos criar os mesmos botões: Ola e adiciona painel
	var btnOla:Button = new Button();
	//definimos as propriedades
	btnOla.x=39.5;
	btnOla.y=108;
	btnOla.label="Olá";
	btnOla.id="botaoOla";
	/*adicionamos um Event Listner, algo novo que nunca falei antes, mas como
	o botão não aceita a definição da propriedade .click em action script
	temos que lhe adicionar um "espia" para disparar a função aoClickarEvent
	assim que for clickado pelo rato (MouseEvent.CLICK). no final iremos ver
	como construir esta função aoClickarEvent(). Se repararem  ao esceverem
	btnOla.addEventListner apareceram variados tipos de eventListners para
	adicionar ao botão, esses  event listners são as maneiras possiveis de
	controlar as acções do utilizador nesse botão e chamr funções consoante a
	acção.
	*/
	btnOla.addEventListener(MouseEvent.CLICK, aoClickarEvent);
	//agora vamos adicionar este botâo como filho (child) do nosso painel2
	//ueremos o botão dentro desse painel.
	painel2.addChild(btnOla);

	//#########################################################
```

```
        //vamos criar o botão adiciona Paniel
        var btnAdd:Button = new Button();
        //definimos as propriedades
        btnAdd.x=92.5;
        btnAdd.y=108 ;
        btnAdd.label="Adiciona Painel";
        //adicionamos um event listner no caso do utilizador clicar
        btnAdd.addEventListener(MouseEvent.CLICK, addPanelEvent);
        //adicionamos ao painel como child.
        painel2.addChild(btnAdd);
    //#########################################################

}
```

E temos a função criaTudo() que irá criar o nosso painel e os 2 botões, falta-nos agora criar as funções que serão chamadas pelos evenlistners que temos em cima, estas funções serão praticamente iguais às que foram criadas no inicio deste livro no ficheiro **olaMundo.as** mas com uma pequena alteração, a função tem que receber dados de um event, e por isso as funções neste **mainScript.as** ficarão assim:

```
private function aoClickarEvent(event:MouseEvent):void {
        Alert.show("Mundo");
}

private function addPanelEvent(event:MouseEvent):void {
        var novo:Panel = new Panel;
        var efeito:Effect = new WipeDown;
        novo.width=180;
        novo.height=115;
        novo.x=0;
        novo.y=0;
        novo.id="panel3";
        efeito.target=novo;
        this.addChild(novo);
        efeito.play();
}
```

Nas funções em cima estão declarados um event:MouseEvent porque foi um listner com um evento/Acção do rato que chamou essa função. Estes **AddListnerEvent** são muito uteis já que nos permitem saber determinadas acções e chamar funções em certos elementos, como ROLL_OVER, ROLL_OUT, MOVE, RESIZE, DRAG, DROP e muitas mais funções.

Agora voltando ao nosso código MXML do olaMundo.mxml, vamos apagar tudo e deixar apenas o seguinte:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="609 height="393" creationComplete="criaTudo();">

<mx:Script source="mainScript.as" />

</mx:Application>
```

Estamos a colocar a nossa aplicação vazia, sem qualquer elementos, e chamámos apenas o script para importar as nossas funções para criar o painel e botões. Mas agora para que o painel e os botões sejam criados temos que chamar a nossa função **criaTudo()**, mas como não queremos juntar mais nenhum botão para chamar essa função, e queremos que ela seja logo executada assim que a aplicação arranca, colocamos na tag **<mx:Application>** o parâmetro **creationComplete="criaTudo()"** este parâmetro indica à aplicação que quando estiver "iniciada" e a sua "criação completa" vai chamar a função criaTudo();, função esta que vai criar o nosso Painel juntamente com os botões...

Salvem a vossa aplicação e corram-a, se clicarem nos botões, estes devem estar a funcionar como se tivessem sido criados por via de MXML, mas na realidade foi tudo criado por via de action script.

6. Efeitos e EventListners

Existem dezenas de efeitos possíveis no flex, embora o flex não tenha sido criado com grandes potencialidades de animação, o mesmo às vezes chega a surpreender pela simplicidade de criação de efeitos e animações personalizadas, em baixo vamos falar de alguns efeitos, eventListners e de como aplicar EventListners a variados componentes.
Vamos pegar no nosso exemplo criado em cima, deixando estar tudo como está, e vamos acrescentar ao código, a seguir ao <mx:script>, um painel (**painel3**):

```
<mx:Panel x="372" y="48" width="227" height="272" layout="absolute" id="painel3"
title="DestinoEfeitos">
      <mx:DataGrid x="10" y="10" width="187" id="meusDados" >
            <mx:columns>
                  <mx:DataGridColumn headerText="Column 1" dataField="col1"/>
                  <mx:DataGridColumn headerText="Column 2" dataField="col2"/>
                  <mx:DataGridColumn headerText="Column 3" dataField="col3"/>
            </mx:columns>
      </mx:DataGrid>
      <mx:Button x="10" y="183" label="Efeito 1" width="89" id="efeito1"/>
      <mx:Button x="107" y="183" label="Efeito 2" width="90" id="efeito2"/>
</mx:Panel>
```

Painel este com uma dataGrid (**meusDados**), 2 botões de efeitos (**efeito1** e **efeito2**), e vamos aproveitar já para encher com dados a nossa dataGrid com um arrayCollection, que funciona com se fosse uma matriz de dados.

Voltando ao **mainScript.as** vamos escrever a seguir ao "imports" e antes/fora de qualquer função, o seguinte:

```
import mx.collections.ArrayCollection;

[Bindable]
public var dadosDataGrid:ArrayCollection = new ArrayCollection([{campo1: "10",
campo2: "12.5", campo3: "0.025"}, {campo1: "1", campo2: "2.5", campo3:
"0.115"},{campo1: "5", campo2: "1.5", campo3: "0.005"}]);
```

E assim temos o nosso fornecedor de dados para a nossa dataGrid, bas agora na dataGrid indicar que estes são os dados a apresentar, Colocamos a instrução **[Bindable]** antes porque queremos/indicamos que este array pode alterar, e se isso acontecer os objectos dependentes dele deverão ser actualizados. **dataProvider="{dadosDataGrid}"**
Voltamos ao nosso mxml, e na tag <mx:DataGrid> colocamos a tag **dataProvider="{dadosDataGrid}"** onde os { } devem ser usados para que se houver alguma alteração nos dados a tabela refrescar os seus dados também, e alterando também nas linhas da tabela <mx:DataGridColumn> o dataField para campo1, campo2, campo3 respectivamente, já que são os campos definidos no array de dados.

Ficará algo como:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="609" creationComplete="criaTudo()" height="393">

	<mx:Script source="mainScript.as" />
<mx:Panel x="372" y="48" width="227" height="272" layout="absolute" id="painel3"
title="DestinoEfeitos">
	<mx:DataGrid x="10" y="10" width="187" id="meusDados"
dataProvider="{dadosDataGrid}">
		<mx:columns>
			<mx:DataGridColumn headerText="Column 1" dataField="campo1"/>
			<mx:DataGridColumn headerText="Column 2" dataField="campo2"/>
			<mx:DataGridColumn headerText="Column 3" dataField="campo3"/>
		</mx:columns>
	</mx:DataGrid>
	<mx:Button x="10" y="183" label="Efeito 1" width="89" id="efeito1"/>
	<mx:Button x="107" y="183" label="Efeito 2" width="90" id="efeito2"/>
</mx:Panel>
</mx:Application>
```

E se correrem agora a aplicação já tem a dataGrid preenchida com dados.

Vamos agora criar um efeito via MXML para animar o aparecimento da data grid, colocamos a seguir ao </mx:Panel> o seguinte:

```
<mx:WipeLeft duration="1000" id="deLado" target="meusDados" />
<mx:Resize target="{[efeito1, efeito2]}" duration="1000" heightBy="20"
id="resizeo" />
```

Temos 2 efeitos, um com o target para a dataGrid **meusDados** e o outro resize para os 2 botões de efeitos. Usamos o **creationCompleteEffect** nos componentes para que quando acabarem de ser criados executem os efeitos, efeitos que são identificados com o **Id**, neste caso tempos o id **deLado** para o WipeLeft e o **resizeo** para o Resize.

Acrescentamos então o creationCompleteEffect na dataGrid com o efeito "deLado" nos 2 botões de efeitos com o efeito "resizeo" e ficaríamos algo como o seguinte:

```
<mx:Panel x="372" y="48" width="227" height="272" layout="absolute" id="painel3"
title="DestinoEfeitos">
            <mx:DataGrid x="10" y="10" width="187" id="meusDados"
creationCompleteEffect="{deLado}" dataProvider="{dadosDataGrid}">
                  <mx:columns>
                  <mx:DataGridColumn headerText="Column 1" dataField="campo1"/>
                  <mx:DataGridColumn headerText="Column 2" dataField="campo2"/>
                  <mx:DataGridColumn headerText="Column 3" dataField="campo3"/>
            </mx:columns>
      </mx:DataGrid>
      <mx:Button x="10" y="183" label="Efeito 1" width="89" id="efeito1"
creationCompleteEffect="{resizeo}" />
      <mx:Button x="107" y="183" label="Efeito 2" width="90" id="efeito2"
creationCompleteEffect="{resizeo}" />
      </mx:Panel>
```

Se correrem a aplicação versão os efeitos na datagrid e nos botões.

Vamos regressar ao mainScript.as e vamos adicionar na função criaTudo, antes do } final, o seguinte:

```
meusDados.addEventListener(DataGridEvent.HEADER_RELEASE, playEffectEvent);
```

e no mesmo ficheiro, fora de qualquer função, vamos criar uma função como aqui apresento:

```
private function playEffectEvent(event:DataGridEvent):void {
      deLado.play();
}
```

e no topo, a seguir aos imports, adicionar:

```
import mx.events.DataGridEvent;
```

Adicionamos um eventListner para chamar a função **playEffectEvent** se o cabeçalho da dataGrid fosse "released" (quando clicamos e largamos o botão do rato), e na função chamamos o efeito que criamos antes e iniciamo-lo: **DeLado.play();**

Vamos adicionar mais um eventListner na função criaTudo() a seguir ao que criamos, algo como:

```
meusDados.addEventListener(ListEvent.ITEM_CLICK, itemListEvent);
```

e no topo a seguir ao ultimo import:

```
import mx.events.ListEvent;
```

e por final, criamos outra função:

```
private function itemListEvent(event:ListEvent):void {
 Alert.show("Item clickado!");
}
```

Por final, se correrem a vossa aplicação, alem dos efeitos iniciais, são executados também os efeitos deLado ao clicar no titulo das colunas da dataGrid, e também é apresentado um alert chamado pelo evento click na lista.

Recapitulando os ficheiros criados:

olaMundo.mxml:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
layout="absolute" width="609" creationComplete="criaTudo()" height="393">
<mx:Script source="mainScript.as" />
<mx:Panel x="372" y="48" width="227" height="272" layout="absolute"
id="painel3" title="DestinoEfeitos">
      <mx:DataGrid x="10" y="10" width="187" id="meusDados"
creationCompleteEffect="{deLado}" dataProvider="{dadosDataGrid}">
            <mx:columns>
                  <mx:DataGridColumn headerText="Column 1"
dataField="campo1"/>
                  <mx:DataGridColumn headerText="Column 2"
dataField="campo2"/>
                  <mx:DataGridColumn headerText="Column 3"
dataField="campo3"/>
                  </mx:columns>
            </mx:DataGrid>
            <mx:Button x="10" y="183" label="Efeito 1" width="89"
id="efeito1" creationCompleteEffect="{resizeo}" />
            <mx:Button x="107" y="183" label="Efeito 2" width="90"
id="efeito2" creationCompleteEffect="{resizeo}" />
      </mx:Panel>

      <mx:WipeLeft duration="1000" id="deLado" target="meusDados" />
      <mx:Resize target="{[efeito1, efeito2]}" duration="1000"
heightBy="20" id="resizeo" />
</mx:Application>
```

mainScript.as:

```
import flash.events.MouseEvent;
import mx.collections.ArrayCollection;
import mx.containers.Panel;
import mx.controls.Alert;
import mx.controls.Button;
import mx.effects.Effect;
import mx.effects.WipeDown;
import mx.events.DataGridEvent;
import mx.events.ListEvent;

[Bindable]

public var dadosDataGrid:ArrayCollection=new ArrayCollection([{campo1:
"10", campo2: "12.5", campo3: "0.025"}, {campo1: "1", campo2: "2.5",
campo3: "0.115"},{campo1: "5", campo2: "1.5", campo3: "0.005"}]);
```

```
private function criaTudo():void {
      //#########################################################
      //declaramos o painel numa nova variavel
      var painel2:Panel = new Panel();
      //definimos os parâmetros "gráficos" do painel
      painel2.x=107;
      painel2.y=89;
      painel2.width=250;
      painel2.height=200;
      painel2.layout="absolute";
      painel2.title="Teste olaMundo";
      painel2.id="panel2";
      //adicionamos como filho do "mainStage"
      this.addChild(painel2);
      //#########################################################
      //vamos criar os mesmos botões: Ola, muda estado e adiciona painel
      var btnOla:Button = new Button();
      //definimos as propriedades
      btnOla.x=39.5;
      btnOla.y=108;
      btnOla.label="Olá";
      btnOla.id="botaoOla";

/*adicionamos um Event Listner, algo novo que nunca falei antes, mas
como o botão não
      aceita a definição da propriedade .click em action script temos que
lhe adicionar um "espia"
      para disparar a função aoClickarEvent assim que for clickado pelo
rato (MouseEvent.CLICK).
      no final iremos ver como construir esta função aoClickarEvent(). Se
repararem  ao esceverem
      btnOla.addEventListner apareceram variados tipos de eventListners
para adicionar ao botão, esses  event listners são as maneiras possiveis
de controlar as acções do utilizador nesse botão e
      chamr funções consoante a acção.*/

      btnOla.addEventListener(MouseEvent.CLICK, aoClickarEvent);
      //agora vamos adicionar este botâo como filho (child) do nosso
painel2

      //queremos o botão dentro desse painel.

      painel2.addChild(btnOla);

      //#########################################################
      //vamos criar o botão adiciona Paniel
      var btnAdd:Button = new Button();

      //definimos as propriedades
      btnAdd.x=92.5;
      btnAdd.y=108 ;
      btnAdd.label="Adiciona Painel";
```

```
      //adicionamos um event listner no caso do utilizador clicar

      btnAdd.addEventListener(MouseEvent.CLICK, addPanelEvent);
      //adicionamos ao painel como child.
      painel2.addChild(btnAdd);

     //#########################################################
      meusDados.addEventListener(DataGridEvent.HEADER_RELEASE,
playEffectEvent);
      meusDados.addEventListener(ListEvent.ITEM_CLICK, itemListEvent);

}

private function itemListEvent(event:ListEvent):void {

      Alert.show("Item clickado!");

}

private function playEffectEvent(event:DataGridEvent):void {

      deLado.play();

}

private function aoClickarEvent(event:MouseEvent):void {

      Alert.show("Mundo");

}

private function addPanelEvent(event:MouseEvent):void {

      var novo:Panel = new Panel;
      var efeito:Effect = new WipeDown;

      novo.width=180;
      novo.height=115;
      novo.x=0;
      novo.y=0;
      novo.id="panel3";

      efeito.target=novo;

      this.addChild(novo);
      efeito.play();

}
```

7. Componentes e Módulos

O flex builder propõe a criação também de componentes e módulos, estes são uma maneira fácil e rápida de organizar-mos o nosso código para que o mesmo não fique pouco optimizado e para evitar alguns problemas de aproveitamento de código. Falaremos de ambos os assuntos, suas diferenças, vantagens e desvantagens bem como a criação e troca de informação entre eles.

7.1. As diferenças entre componentes e módulos.

Inicialmente as diferenças podem não parecer muitas, mas na realidade faz toda a diferença, no menu do flex builder se clicarem em File -> New conseguem ver uma lista, de entre os quais existem "MXML Module" e também "MXML Component". A grande diferença destes 2 é que o Componente ao ser criado estará disponível para "juntar" ao nosso "stage" principal como elemento drag & drop na janela "Components" na pasta "Custom", e o Módulo não estará acessível dessa forma.

Algumas características dos Componentes:

-Podem ser baseados em componentes já existentes (Painel, Canvas, TitleWindow, ...)
-São controlados facilmente através do nosso "stage" já que passam a estar incorporados no nosso .swf principal
-Aumentam o tamanho do nosso swf final e por consequente afectam o desempenho da nossa aplicação.
-Em aplicações maiores podem causar alguma confusão no código.

Quanto aos componentes, estes são "Aplicativos" independentes ou seja, são criados como se pertencerem à nossa aplicação principal, mas na realidade estão fora dela própria. Cada módulo cria um .swf próprio, que é completamente externo à nossa aplicação, além de poder ser usado em futura aplicações deixa o nosso código bem mais simples, e muito mais facilmente editável e organizado.

Algumas características dos Módulos:
-Leves, rápidos e compactos.
-Não aumentam o tamanho da nossa aplicação principal.
-Podem ser facilmente alterados sem ter que alterar a aplicação principal.
-Facilmente actualizáveis junto da nossa aplicação principal.
-Podem ser re-usados em qualquer outra aplicação.

Um dos grandes problemas para quem não tenha grande conhecimento de action script pode ser a comunicação da nossa aplicação principal com o nosso módulo, já que grande parte da nossa comunicação deverá ser feita através de eventos. Mas mais à frente mostrarei um exemplo de como chamar uma função do nosso módulo e tratar a resposta dessa função na nossa aplicação principal.

7.2. Criando um Componente e trabalhando com ele

Um componente como disse em cima não é nada mais que um componente personalizado, como se se trata-se de por exemplo um painel personalizado logo é tratado como componente na nossa aplicação.

Vamos então criar um Componente, tomando em conta que ainda continuamos a usar a nossa aplicação **olaMundo** que criamos anteriormente, e para isso vamos ao menu do flex builder e escolhemos:

File -> New -> MXML Component

Como "filename" escrevemos **dbConf** e na opção "based on" procuramos na lista o **TitleWindow** e deixamos o resto como está clicando no "finish".
Se tudo correu bem, ter-vos-à aparecido um novo documento (dbConf.mxml) apenas com o código:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="400" height="300">

</mx:TitleWindow>
```

A diferença entre este código e o código que aparece ao criar-mos a nossa primeira aplicação é que este inicia logo com o <mx:TitleWindow> para indicar ao compilador que a sua base será a TitleWindow deverá ser tratado como tal na nossa aplicação principal.

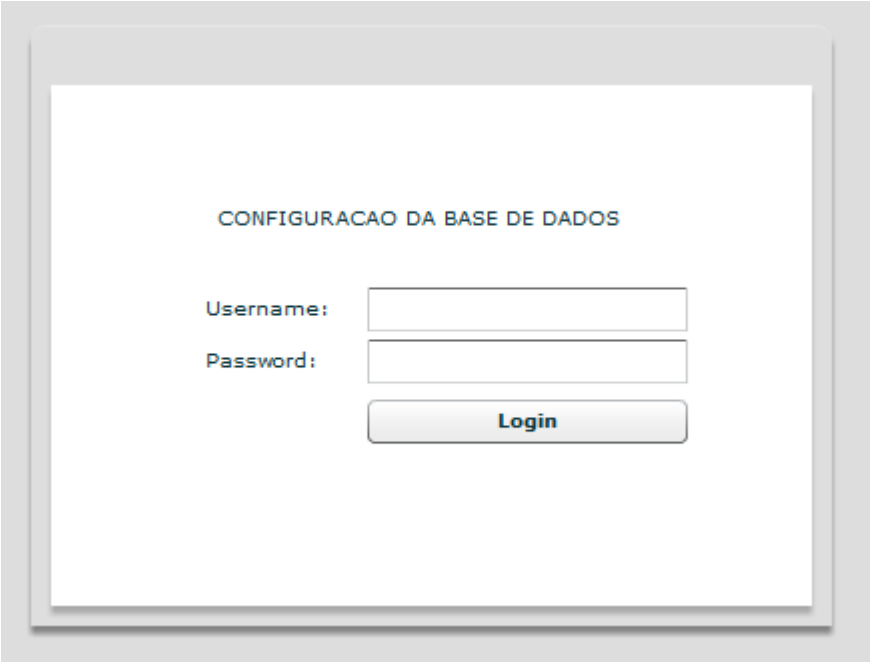
Se clicarem no modo de "design" verão que apenas é apresentado o nosso TitleWindow. Vamos aproveitar já para fazer deste componente o nosso menu para fazer um sistema de login com uma futura ligação a uma base de dados mysql, colocando 3 labels, 2 inputText (um deles como password) e 2 botões.
Fica ao vosso critério a colocação dos labels "titulo", "username :", "password :" e dos campos inputText com os id's respectivamente iguais a "inputUser" e "inputPass". E o nosso botão com o id "btnLogin".

Notem que nas opções do inputText "password" devem seleccionar nas propriedades o "Display as password" colocando-o a "true".

Usando a minha interface e código:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:TitleWindow xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="400" height="300">
	<mx:Label x="80.5" y="58" text="CONFIGURACAO DA BASE DE DADOS"
width="219"/>
	<mx:Label x="75" y="103" text="Username:" width="75"/>
	<mx:Label x="75" y="129" text="Password:" width="80.5"/>
	<mx:TextInput x="158" y="101" id="inputUser"/>
	<mx:TextInput x="158" y="127" displayAsPassword="true" id="inputPass"/>
	<mx:Button x="158" y="157" label="Login" width="160" id="btnLogin"/>
</mx:TitleWindow>
```

Estará algo como:



Guardamos este componente e voltamos ao nosso exemplo olá mundo.
Como a nossa aplicação olá mundo já está a ficar um tudo nada "non-friendly" ou seja, já está a ficar com muitas "inutilidades" vamos começar por eliminar alguns componentes que criámos...

como tínhamos um painel a ser adicionado por action script logo no inicio, vamos limpar um pouco a nossa aplicação; Um dos pontos que não tinha falado anteriormente era como remover o nosso painel criado por via de action script do nosso "stage", mas facilmente conseguiriam tira-lo ao criar um botão que executasse o seguinte código : this.removeChild("id"); onde o id seria o nosso id criado pela função, neste caso this.removeChild("panel3"); ("novo") ou então de uma maneira mais drástica: this.removeAllChildren(); que elimina todos os nosso componentes adicionados com o addChild.

Vamos então desactivar o creationComplete na aplicação onde carregava os nosso outros componentes, (painel olá mundo, botão "olá" e "Adiciona panel"), para isso basta retirar do <mx:Application ... > o "trigger" creationComplete="criaTudo()".

Ficamos apenas com a nossa dataGrid e os nosso efeitos. Vamos aproveitar para adicionar uma ApplicationControllBar que pode ser arrastada para o nosso stage, coloquem-na no topo do stage e adicionem-lhe um botão com o label "Ligar BD" de maneira a que no vosso código fique algo como:

```
<mx:ApplicationControlBar x="10" y="6" width="589">
      <mx:Button label="Ligar BD"/>
</mx:ApplicationControlBar>
```

vamos então ao nosso mainScript.as e fazemos as seguintes actualizações:

no topo a seguir aos nosso imports vamos colocar:

```
import dbConf;
import mx.managers.PopUpManager;
```

e no final a seguir à nossa ultima função a função seguinte:

```
private function abrePainelLogin(centrado:Boolean):void {
	var painel:dbConf = new dbConf();
	painel.showCloseButton=true;
	PopUpManager.addPopUp(painel, this, true);
	if(centrado==true) PopUpManager.centerPopUp(painel);
}
```

e no nosso dbConf.mxml colocamos na nossa tag <mx:TitleWindow ...> o seguinte:

```
close="PopUpManager.removePopUp(this)" removedEffect="fechar"
```

(notem que adicionei um pequeno efeito para fechar (removedEffect) que vamos criar a seguir com o id "fechar") e teremos que fazer o import do PopUpManager senão vamos ter erros... basta adicionar antes do </mx:TitleWindow> o seguinte:

```
<mx:Script>
		<![CDATA[
			import mx.managers.PopUpManager;
		]]>
</mx:Script>
```

de volta ao nosso olaMundo.as no botão que criamos dentro da nossa control bar no "trigger" click chamamos a nossa função (centrada=**true**):

```
<mx:ApplicationControlBar x="10" y="6" width="589">
		<mx:Button label="Ligar BD" click="abrePainelLogin(true)"/>
	</mx:ApplicationControlBar>
```

e vamos criar os nosso efeitos chamados ao fechar o panel, para isso colocando o efeito a seguir à nossa applicationControlBar:

```
<mx:Parallel id="fechar">
		<mx:Zoom />
		<mx:Fade />
	</mx:Parallel>
```

Vamos então a explicações. O que fiz foi importar o nosso dConf.mxml para o nosso stage para ser usado, e importei também o PopUpManager que será usado para fazer do nosso dbConf um popUp.

Criei a função para abrir o nosso dbConf como popup e adicionei um parâmetro que é recebido na função: centrado:Boolean que receberá apenas (true ou false) que serve para vos explicar também como enviar parâmetros para a função ao ser chamada. Dentro dessa função declarei a variável painel como "portador" do nosso dbConf.mxml e adicionei-lhe um botão de fechar:

```
painel.showCloseButton=true;
```

no nosso dbConf.mxml adicionamos o trigger "close" para remover o popup ao fechar no nosso dbConf.mxml e o removedEffect para executar o efeito que criamos ao ser fechado.
O nosso efeito com id="fechar" é um efeito paralelo, ou seja ambos os efeitos Zoom e Fade são executados ao mesmo tempo.
Podem guardar os vosso ficheiros CTRL+SHIFT+S, e corram a aplicação... se carregarem no Login BD aparecerá o nosso dbConf.mxml e se clicarmos no botão de fechar o nosso painel será removido com o efeito zoom+fade ("fechar").
Em si trabalhar com componentes é assim tão simples como importa-los e usa-los.

7.3 Enviando e recebendo dados de/para o nosso componente.

Para comunicarmos com o nosso componente, no caso de queremos enviar dados para ele, torna-se tão simples com o seguinte (suponhamos que no campo user do nosso dbConfig queremos colocar um user predefinido), basta na função "abrePainelLogin" colocarmos o seguinte:

```
painel.inputUser.text="teste";

a seguir ao PopUpManager.centerPopUp();
```

se escreverem painel. Aparece toda a lista de elementos do nosso painel, incluindo labels, buttons e inputs. Para recebermos dados do nosso painel existem duas formas, ou adicionamos um eventListner para detectarmos a instrução de fechar, e acedermos às propriedades antes de remover o popup:

```
painel.addEventListener(CloseEvent.CLOSE, buscaDados);

private function buscaDados(event:CloseEvent):void{

      Alert.show(event.currentTarget.inputUser.text);

}
```

Este tipo de função e evento, não é muito usual e acaba por obrigar à duplicação de código...e apenas é disparado quando o popup é fechado, e por isso não é muito aconselhável em projectos grandes que necessitamos de usar o mesmo componente varias vezes em diferentes objectivos e de saber determinados valores sem estarmos dependentes dos events do componente.

A segunda forma é a mais lógica, que é disparar eventos personalizados e usar o eventListner para os escutar. Vamos por exemplo alertar o user que clicou no botão "Login" do nosso popup dbConfig.

O mais rápido seria colocar um evento no click do botão, mas vamos verificar isso com o nosso stage para termos a certeza que o nosso dbConf está a enviar dados/eventos para o nosso Stage.

Para isso basta criarmos uma função no nosso dbConf a seguir ao

```
import mx.managers.PopUpManager;

private function clickado():void {
this.dispatchEvent(new Event("clickado"));
                  }
```

e vamos chamar esta função no botão login no "trigger" click:

```
<mx:Button x="158" y="157" label="Login" width="160" id="btnLogin"
click="clickado()" />
```

se salvarem e correrem a vossa aplicação agora e chamarem o popup já aparece o campo user com o texto "teste" e se carregarem no login, não acontece nada, mas na realidade ao ser clicado está a despachar o evento "clickado", mas nada acontece porque não o estamos a escutar em lado nenhum. Vamos então ao nosso mainsScript.as e na função "abrePainelLogin" antes do popUpManager.createPopup colocamos:

```
painel.addEventListener("clickado", lidaClickado);
```

e no mesmo mainScript.as criamos a seguinte função:

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

```
		Alert.show("O botão Login do painel dbConf foi clickado!!\nO texto escrito
nos campos\n user: "+event.currentTarget.inputUser.text+"\npass:
"+event.currentTarget.inputPass.text+"\n\nCerto ??");

}
```

salvem os vossos ficheiros e corram a aplicação. Testem o botão Login com diferentes valores nos inputs user e pass.
Como podem ver podemos criar eventos onde quisermos e sempre que quisermos, esta forma do dispatch event, é muito útil porque podemos usar o nosso componente onde quisermos e sem ter que o estar a alterar para adaptar a uma outra aplicação.

7.4 - Criando um Módulo e trabalhando com ele

A nível de módulos o sistema funciona quase da mesma forma que um componente como foi ditem em cima, excepto que no caso de módulos a comunicação entre o módulo e a nossa aplicação é mais elaborada. Vamos criar um módulo com o mesmo componente que criamos antes (dbConf.mxml):

File->New->MXML Module

vamos dar-lhe o nome de modLogin e deixar os tamanhos como (250*200) e ao fundo vemos a opção de optimização, neste caso vamos escolher "Do not optimize..." para que possamos depois testar em outra aplicação que não o nosso olaMundo.
Abre-se então o nosso recem-criado módulo:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Module xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="250" height="200">

</mx:Module>
```

que funciona exactamente igual ao nosso dbConf ou olaMundo, com modo design activo também.
Vamos então copiar o contudo do código do dbConf.mxml (exceptuando o <mx:TitleWindow> e </mx:TitleWindow>) e colar dentro do nosso modLogin:

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Module xmlns:mx="http://www.adobe.com/2006/mxml" layout="absolute"
width="250" height="200">
	<mx:Label x="15.5" y="10" text="CONFIGURACAO DA BASE DE DADOS"
width="219"/>
	<mx:Label x="10" y="54" text="Username:" width="75"/>
	<mx:Label x="10" y="93" text="Password:" width="80.5"/>
	<mx:TextInput x="80" y="52" id="inputUser"/>
	<mx:TextInput x="80" y="91" displayAsPassword="true" id="inputPass"/>
	<mx:Button x="80" y="139" label="Login" width="160" id="btnLogin"
click="clickado()" />
	<mx:Script>
		<![CDATA[
			//import mx.managers.PopUpManager;

			private function clickado():void {
				this.dispatchEvent(new Event("clickado"));
			}
		]]>
	</mx:Script>
</mx:Module>
```

e vamos apagar/comentar a linha import mx.managers.PopUpManager; já que não necessitamos dela para nada neste caso. (Notem que eu alterei um pouco posições relativas (x,y) do nosso mx:Module para caber no nosso stage.)

Guardamos e o nosso módulo está criado, bem simples, vamos agora de volta ao nosso olaMundo e vamos inserir no código um painel, um ModuleLoader e uma ProgressBar (ambos drag & drop) mas por questões de ficar com o layout igual ao meu basta colarem o seguinte no código do olaMundo antes da tag </mx:Application>:

```
<mx:Panel x="10" y="47" width="354" height="310" layout="absolute">
            <mx:ModuleLoader x="10" y="10" width="314" height="226" id="loader">
            </mx:ModuleLoader>
            <mx:ProgressBar x="124" y="240" id="loaderBar"/>
      </mx:Panel>
```

defini também o id da progressBar como "loaderBar" e do nosso ModuleLoader como "loader". Vamos adicionar o seguinte botão à nossa applicationBar a seguir ao nosso botão Ligar BD:

```
<mx:Button label="Carrega Modulo" click="carregaModulo()"/>
```

e no nosso mainScript.as a seguinte função:

```
private function carregaModulo():void{

      loader.url="modLogin.swf";
      loader.loadModule();
      loaderBar.source=loader;
}
```

aqui indicamos que o modulo a carregar no loader é o modLogin.swf (depois de a aplicação compilada no nosso modLogin é compilado num swf à parte com o nome de modLogin.swf) e que a progressBar "loaderBar" deve ter como source o nosso moduleLoader "loader" para acompanhar o seu carregamento.

Salvem e corram, cliquem no Carrega Modulo e verão o vosso modulo a ser carregado bem como o progresso do carregamento na ProgressBar, como no final do módulo e antes desse carregamento a progressBar deixa de ter sentido, vamos oculta-la antes e depois do carregamento, mostrando-a apenas durante o carregamento do módulo. Para isso vamos colocar na tag <mx:ProgressBar ..> o seguinte: `visible="false"`

e na nossa função carregaModulo:

```
loaderBar.visible=true;
```

e para a ocultarmos, deveremos ainda colocar um eventListner para sabermos quando o módulo já foi carregado para que então possamos ocultar a nossa progressBar:

```
loader.addEventListener(ModuleEvent.READY, loadTerminado);
```

e criar a função para ocultar:

```
private function loadTerminado(event:ModuleEvent):void{
      loaderBar.visible=false;
      loader.removeEventListener(ModuleEvent.READY, loadTerminado);

}
```

salvem e corram o vosso aplicativo, agora apenas é apresentada a barra da progressBar enquanto o módulo está a ser carregado.

Quanto a acedermos e enviarmos dados para o nosso módulo, as coisas ficam um pouco mais complexas do que um componente.

Para receber dados ou obter dados, podemos usar o event dispatcher que criamos com o event "clicado", que vamos já passa a testar.

Na função loadTerminado vamos adicionar o seguinte:

```
loader.child.addEventListener("clickado", lidaClickado);
```

e aproveitamos a mesma função lidaClickado criada anteriormente, se correrem a aplicação verão que funciona na mesma.

Para enviarmos dados ou definir dados no módulo, temos duas formas, se os dados a receber form cruciais à aplicação, ou seja, forem sempre necessários,o que é aconselhado por uma questão de reutilização de código, é criarmos uma função no módulo que recebe o parâmetro/texto a definir. Se por outro lado a informação a receber no módulo for apenas esporádica, podemos aceder directamente ao componente, ou elemento que queremos afectar.
Vou passar a explicar as duas formas, tomando por exemplo que queremos colocar dados no campo inputUser do nosso modulo após o seu carregamento.

Vamos criar uma função no nosso modLogin, a seguir á função que despacha o nosso evento "clickado", da seguinte maneira:

```
 private function defineUser(user:String):void{

                  inputUser.text=user;
}
```

voltamos então ao nosso mainScript.as a seguir à função loadTerminado e criamos a seguinte função:

```
private function define():void{
      if((loader.getChildren()).length>0) (loader.child as
modLogin).defineUser("TESTANDO");
      else Alert.show("Modulo ainda Não carregado");
}
```

esta função verifica se o nosso loader (ModuleLoader) já possui algum "child", ou seja, se o nosso modLogin já está disponível para ser utilizado, e caso esteja chama a nossa função defineUser do módulo.

Notem que ao colocar loader.child as modLogin estamos a indicar que o child/modulo carregado é uma "copia" do nosso modLogin para que o child do loader seja disponibilizado/utilizado com as funções e componentes do modLogin.

Para aceder ou definir qualquer dado do nosso module, este deve ser sempre tratado como disse em cima.

Esta é uma forma de passar parâmetros através de uma função, vamos então ve-la em acção, para isso vamos criar um botão no nosso olaMundo.mxml ao fundo do nosso painel, ao lado da progressBar "invisível" e em baixo do nosso moduleLoader.

Mais uma vez por questão de layout será melhor copiarem o código:

```
<mx:Button x="10" y="244" label="Define user" click="define();"/>
```

e colocarem-no em baixo a seguir ao:

```
<mx:ProgressBar visible="false" x="124" y="240" id="loaderBar"/>
```

Salvem a vossa aplicação e testem, se clicarem no botão "Define user" sem o modulo estar carregado vão receber um erro "Modulo ainda não carregado", mas se carregarem o módulo e voltarem a clicar nesse mesmo botão, o textInput do user passará a ser definido como "TESTANDO".

A outra possibilidade sem vez de utilizar uma função do módulo será aceder directamente ao campo inputUser e definir o seu texto, bastando substituir na função que criamos o chamar da função pelo campo, definindo aí o seu valor (.text). Ficaria algo como:

```
private function define():void{
      if((loader.getChildren()).length>0) (loader.child as
modLogin).inputUser.text="TESTANDO"
      else Alert.show("Modulo ainda Não carregado");
}
```

e teriam o mesmo efeito, mas a aceder directamente ao campo. Se necessitarem de usar bastantes vezes dados/funções do modulo, é altamente aconselhável definirem uma variável global para guardar o vosso modulo, para evitar estarem sempre a escrever (loader.child **as** modLogin), algo como definir no nosso mainScript.as, a seguir aos imports o seguinte:

```
public var modulo:*;
```

e na função loadTerminado colocarem:

```
modulo=(loader.child as modLogin);
```

e aproveitamos para testar, substituindo na função define o (loader.child **as** modLogin) por modulo e ficariamos com algo como:

```
private function define():void{
      if((loader.getChildren()).length>0) modulo.inputUser.text="TESTANDO"
      else Alert.show("Modulo ainda Não carregado");
}
```

desta feita, podemos aceder ao nosso modLogin em toda a aplicação (olaMundo) através da variável modulo.

O trabalho com módulo pode passar por ser mais difícil, mas na hora da publicação da vossa aplicação e principalmente na hora da reutilização dos mesmos componentes por outras aplicações, torna-se muito mais simples e útil, já que nos pouca dezenas, se não, centenas de linhas de código além de que o nosso módulo pode ser distribuído e utilizado por outros utilizadores em outras aplicações de uma maneira simples e rápida.

8. Entendendo a comunicação com Objectos Remotos (php/mysql) via amfphp

O flex em si ainda não está completamente implementado com outras linguagens de programação, e embora já se possa criar uma aplicação com acesso a uma base de dados (Data->Create Application from database) o seu potencial é um bocado limitado comparando com as possibilidade de uma linguagem como o php, e como por vezes precisamos de dados de uma linguagem "server-side" aí o flex não tem mesmo qualquer potencialidade...
Por este e por muitos outros factores, existem vários "aplicativos" que servem de interface para que o flex possa comunicar com praticamente qualquer linguagem server side, tais como:

- PYamf, para Python
- AmfPHP para PHP
- FabreAMF para PHP5
- BlazeDS para Java
- Red5 para Java
- Fluorine para .Net

Basicamente o que estes sistemas fazem é transformar os pedidos do Action Script e transformarem-nos em pedidos para a linguagem destino, e devolver os resultados desse pedido à linguagem para um formato que o Action Script possa entender através de um processo chamado serialização.

Cada um dos sistema em cima necessita de algumas configurações que podem encontrar junto do download ou na pagina desses mesmos sistemas.

8.1. Instalando o amfPHP e Wamp Server

O que vou falar chama-se amfPhp. Devem então fazer o download do mesmo em
e podem também encontrar no site ofical (http://www.amfphp.org/) alguma documentação bem como alguns exemplos e tutoriais.
Depois do download feito, basta extraírem o conteúdo do arquivo (.zip) para qualquer directoria e está pronto a usar. É bem simples o seu funcionamento como vamos ver mais à frente. Agora vamos proceder às configurações necessárias para poder ter a correr no nosso olaMundo um sistema que nos permita trabalhar com o php e por consequente o mysql.

Vamos copiar então a pasta "amfphp" para dentro da pasta do nosso projecto e de seguida abrir essa mesma pasta, se seguida abrir a pasta "services" onde podem encontrar um ficheiro que diz "place_services_here", ora estamos no sitio certo para criar o nosso serviço.

Um serviço é como se tratasse de uma interface dentro do próprio amfphp, é também o serviço que será identificado no flex.
Notem que para isto funcionar necessitam de ter o vosso projecto (olaMundo) dentro de um "servidor" web que tenha suporte a php. Existem alguns bastante simples de instalar e de colocar a funcionar, tal como o WAMP (http://www.wampserver.com/), bastando depois instalar com os parâmetros por defeito e escolher o user e password da base de dados do mysql (vamos usa-los de seguida)

Este processo de trocar o nosso projecto (olaMundo) para uma directoria diferente da actual é mais complicado do que apenas mudar a localização, já que o flex não gosta muito disso e ainda por cima devemos trocar o nosso projecto para trabalhar com um servidor. O que aconselho, supondo que

instalamos o wamp em c:/wamp é fazer o seguinte. Salvamos todos os nossos ficheiros do projecto (olaMundo) e fechamos o projecto (botão direito->Close Project).
Vamos depois copiar a pasta do nosso projecto para o servidor em c:\wamp\www\

Voltamos ao flex e no final do projecto fechado temos que o apagar (botão direito->Delete) e escolher "Do not delete Contents" já que o novo projecto a importar terá o mesmo nome.
No final vamos ao menu File->New->Flex Project.

No nome colocamos olaMundo, em projecto location, clicamos na "checkbox" para alterar e colocamos c:\wamp\www\olaMundo e em baixo escolhemos Application Server Type -> PHP

Clicamos em Next e deveremos colocar (se não foi colocado automaticamente)
Web root: c:\wamp\www\olaMundo
Root Url: http://localhost/olaMundo

Output Folder: bin-debug

clicamos em Validate Configuration e de novo em Next
em Main source folder: (aqui devemos verificar se na pasta olaMundo do nosso c:\wamp\www\olaMundo o olaMundo.mxml está na raiz da pasta ou na pasta src, já que na altura da criação do nosso pojecto anterior eu defini o main source folder sem valor e logo foi criado o olaMundo.mxml na raiz, se usaram os valores por defeito na altura da criaçao devem deixar estar como "src")

Vamos supor que está na raíz, e limpamos este campo, em Main Application File devemos ter o olaMundo.mxml, clicamos em finish e o flex actualizará o nosso projecto e está pronto para usar de novo, mas desta feito no nosso servidor.

8.2. Criando o primeiro serviço no AmfPHP

Voltando à pasta c:\wamp\www\olaMundo\amfphp\services\
Vamos criar uma pasta chamada **ola** e dentro dela criar um ficheiro de texto com o nome de **mundo** e guarda-lo com a extensão php, se o ficheiro não for guardado como .php deve aparecer o nome como mundo.php e ao abrir vi abrir no bloco de notas (é sinal que o Sistema Operativo está a ocultar extensões), neste caso, apaguem o ficheiro (mundo.php.txt) e voltem à pasta services, abram a pasta amfphp e copiem o ficheiro que ela contem (DiscoveryService.php) e copiem-no para a pasta ola, depois mudem o nome para mundo e abram esse ficheiro no bloco de notas e apaguem todo o seu conteúdo e escrevam apenas o seguinte a titulo de exemplo:

```
<?php
class mundo
{
	function teste($nome) {

	return "Olá ".$nome;

	}
	function mundo() {
	//Função principal do serviço, que poderá guardar variáveis
	//sempre disponíveis ao longo das nossas funções, tal como
	//uma ligação à base de dados
	}
	}
?>
```

e guardem o ficheiro.

O amfphp tem um browser que permite escutar os nossos serviços e testa-los, logo vamos aproveitar para testar o serviço que criamos. Acedam com o vosso browser de internet a: http://localhost/olaMundo/amfphp/broswer e vai-se abrir uma pagina a pedir informações sobre o gateway, que devem deixar estar como está por defeito clicando em save.
Do vosso lado esquerdo deve aparecer os serviços amfphp e o nosso recem criado ola, cliquem em cima dele e escolha o mundo. Do lado direito deve aparecer apenas a função "teste" com um campo a pedir a nossa variável $nome, vamos introduzir o vosso nome, exemplo joão, e carregar em call.

Se tudo correr bem ao fundo dessa pagina terão algumas tabs e na Results aparecerá o devolvido pelo return da função, neste caso olá joão, provavelmente irão aparecer uns caracteres esquisitos, já que o valor é devolvido com um padrão de caracteres diferente, para alterar isso voltamos ao nosso mundo.php e colocamos em vez de return "olá ".$nome colocamos:

```
return "Olá ".utf8_decode($nome);
```

e testem o call de novo no browser. Agora deve devolver em Results a string "Olá joão" correctamente.

Neste momento o serviço está a funcionar correctamente, só temos que fazer com que o flex o teste com sucesso também.

8.3. Configurando o Flex Builder para trabalhar com o amfPHP

O flex por si não sabe o que é o amfphp e muito menos onde o ir buscar, por isso deveremos incluir um services-config.xml na altura da compilação.

Podem fazer o download do meu blog do services-config.xml já configurado para a nossa aplicação.

Download em: http://www.msdevstudio.com/mywork/services-config.rar

devem extrair o ficheiro services-config.xml para a raiz do projecto (ou na pasta src conforme definido antes no main source folder) (c:\wamp\www\olaMundo) e caso o caminho da aplicação no vosso servidor não seja o http://localhost/olaMundo devem editar este services-config.xml com o bloco de notas e ao fundo onde tem:

```
uri="http://localhost/olaMundo/amfphp/gateway.php"
```

devem alterar para o caminho correcto do vosso servidor e guardar.

Depois voltamos ao flex, ao nosso projcto olaMundo e vamos ao menu:
Project->Properties->Flex Compiler e no campo onde tem -locale en_US vamos escrever:

```
-locale en_US -services "services-config.xml"
```

e carregar em Apply, aguardem a actualização e cliquem em Ok.

Se aparecer um erro, cliquem em cima do projecto OlaMundo com o botão direito e depois Refresh e o erro deve desaparecer.

8.4. Criando o primeiro Remote Object no Flex

Supondo que tudo correu bem, estamos prontos para o nosso primeiro teste com objectos remotos, neste caso o amfPhp, para isso no código do nosso olaMundo.mxml antes da tag </mx:Application...> colocamos os nosso serviço:

```
<mx:RemoteObject id="nossoObjecto" destination="amfphp" source="ola.mundo">
        <mx:method name="teste" result="{lidaTeste(event)}">
                <mx:arguments>
                        <nome>
                                {"António"}
                        </nome>
                </mx:arguments>
            </mx:method>
      </mx:RemoteObject>
```

e criar a seguinte função no nosso mainScript.as no final:

```
private function lidaTeste(evento:ResultEvent):void {
      Alert.show("Adivinha? o php respondeu à nossa chamada e
devolveu:\n"+(evento.result as String));
}
```

e colocar junto dos imports:

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

O nosso Objecto está pronto, e a função para lidar com os resultados também... vamos adicionar um botão ao lado no nosso "Define user" no olaMundo.mxml, bastando no código a seguir a esse botão colocar:

```
<mx:Button x="110" y="244" label="Chama RO" click="nossoObjecto.teste.send()"/>
```

Guardem todos os vossos ficheiros vamos a explicações antes de lançar a aplicação; Criamos o nosso RemoteObject com o id="nossoObjecto" que será utilizado para chamar qualquer função deste objectos, com o destination="amfphp" que é indicado no services-config.xml e com o source="ola.mundo" que são o nosso serviço:

pasta.ficheiro dentro da pasta services do amfphp.

Depois temos o <mx:method ..> onde o name é exactamente o nome da nossa função php, para cada função devemos ter um método.

O result é a função que vai lidar com os resultados devolvidos pelo php. A tag <mx:arguments> indica o inicio dos argumentos a passar, no nosso caso a função do php teste apenas recebe a variável $nome (<nome>) e no nosso código colocamos o conteúdo da variável, neste caso "António".

Se tivermos mais variáveis a receber, devemos coloca-las sempre pela respectiva ordem... um exemplo, se a função php receber 3 variáveis, por exemplo:

function teste($nome, $idade, $local); devemos colocar no nosso remote object o seguinte:

```
<mx:arguments>
	<nome>
		{"António"}
	</nome>
	<idade>
		{"21"}
	</idade>
	<local>
		{"Lisboa"}
	</local>
</mx:arguments>
```

Logo depois temos a nossa função para "tratar" os dados recebido no remoteObject, a qual recebe um evento de resultado (evento:ResultEvent) no qual são incluídos os dados enviados pelo amfphp tratados como evento.result, ao fazer o Alert.show(); tratei-os como string (as String)

Basta então correrem a vossa aplicação depois de a ter salvo e carregarem no botão "Chama RO" e se tudo correu bem, têm a vossa aplicação a comunicar com o php via amfphp remote object.

Bem, por agora é tudo, brevemente vamos continuar com o acesso a uma base de dados mysql e ir buscar dados para preencher a nossa datagrid se o login for feito com sucesso.