

Instant Flex on Rails



Uma nova forma de pensar e desenvolver na web

Por: Carlos Eduardo
Empresa: e-Genial Soluções Inteligentes
Site: <http://www.egenial.com.br>
e-mail: carlosetuardo@egenial.com.br

1	Introdução.....	3
2	Mais o que é RIA?	3
3	Breve introdução sobre Flex?.....	3
4	O que vamos praticar?	3
5	Instalação	4
6	Hello Word e-Genial Flex	5
7	Iniciando o corpo da aplicação.....	6
8	ActionScript.....	6
9	Criando uma window.....	7
10	Criando os elementos campos e botão.....	8
11	Compilando e testando	9
12	Criando uma simples aplicação Rails.....	11
13	Criando a camada de visualização com Flex	15
14	Como aprender mais?	22

1 Introdução

A internet é sem dúvida, uma ótima ferramenta para encontrar e organizar informações, mas no que se refere a aplicações de usabilidade, interatividade em tempo real e multimídia, está apenas começando a dar seus primeiros grandes passos graças a WEB 2.0, é neste ponto que você vai entender e conhecer o que é RIA e de cara aprender os principais passos para integrar o Adobe Flex ao Ruby on Rails.

2 Mais o que é RIA?

O RIA é um conceito inovador no modo de pensar e desenvolver na web. Uma aplicação RIA une a funcionalidade dos softwares para computadores de mesa com o extenso alcance e facilidades econômicas de aplicativos para internet, o que proporciona um novo nível de experiências para usuários e desenvolvedores.

Atualmente no mercado as principais ferramentas para o desenvolvimento de RIA são o Openlaszlo, Flex e o Flash da Adobe. O flash só é capaz consultar bases de dados ou fazer qualquer operação no servidor com o auxílio de algum recurso externo, ou seja, tecnologia Remoting, já o Openlaszlo e o Flex possuem recursos internos baseados em SOAP para a comunicação com o servidor.

3 Breve introdução sobre Flex?

Flex é uma tecnologia voltada para aplicações RIA. No Flex usamos uma linguagem de marcação, o MXML que é baseada no XML, para definir a interface da aplicação e o Actionscript 3.0 para a parte lógica. As aplicações Flex levam a extensão `.mxml` e podem ser criadas em qualquer editor de texto comum, como o Bloco de Notas.

4 O que vamos praticar?

Em nosso exemplo vamos criar uma pequena aplicação Flex que vai adicionando dados em uma Grid, vamos enviar o valor digitado em um campo e em seguida persistir na sessão, deletar todos ou cada item adicionado na Grid, isso tudo realizando toda a comunicação via SOAP, e para cuidar das nossas regras de negócios vamos usar o magnífico [Ruby on Rails](#).

5 Instalação

Partimos do princípio que já tenha o Ruby on Rails instalado em sua máquina e funcionando corretamente, mas, caso não tenha poderá encontrar em meu blog www.blog.egenial.com.br, o pdf que ensina o passo a passo da instalação.

No exemplo vamos usar o Flex 2.0 SDK, uma versão Free do Flex que inclui as bibliotecas e classes do Flex e o compilador MXML e Actionscript 3.0, a única diferença em comparação ao Flex Builder 2.0 é que no Flex SDK a compilação do projeto/arquivo mxml é por linha de comando, por ser Free* a Adobe deu um grande passo a favor da comunidade, que na verdade vejo com uma forma de chegar mais junto e presente ao mercado, então liberaram a licença das aplicações geradas pelo Flex, ou seja, você poderá criar aplicações extremamente comerciais, e, até mesmo distribuí-las de graça. Já o Flex Builder tem todo o ambiente e poder do SDK + uma IDE muito boa baseada no eclipse para criar e gerar aplicações Flex totalmente visuais do tipo clica arrasta e solta, bom o nosso primeiro passo é realizar o download do Flex SDK no link a seguir:

<https://www.adobe.com/cfusion/tdrc/index.cfm?product=flex>

Em seguida vai pedir para você se registrar, faça o breve cadastro e realize o download do seguinte arquivo:

Free Flex 2™ Software Development Kit

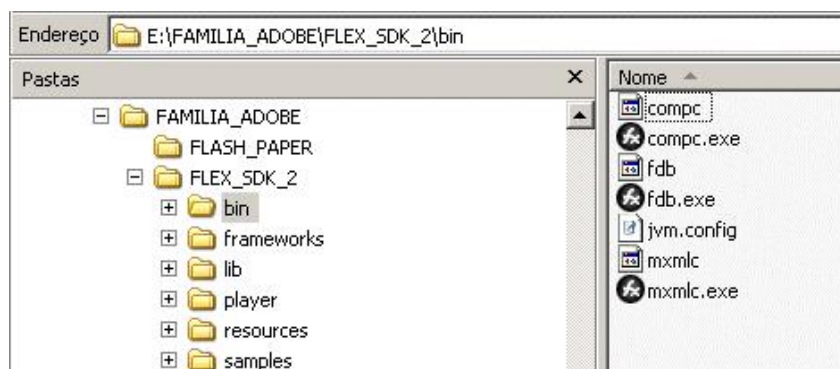
The Flex 2 SDK includes the Flex framework (component class library) and Flex compiler, enabling you to freely develop and deploy Flex applications using an IDE of your choice (already included with Flex Builder 2).

 Flex SDK

All | All Platforms | 27.98 MB

Download

Agora extraia o arquivo compactado em um lugar de fácil acesso ex: c:\flex



No meu caso extrai em e: \família_adobe\flex_sdk_2

6 Hello Word e-Genial Flex

Bom agora que já temos o ambiente e as bibliotecas do flex no seu devido lugar, vamos realizar um pequeno teste no estilo do velho "Hello Word e-Genial".

Crie um arquivo com o nome **Helloworld.mxml** dentro da pasta bin mesmo somente para adiantarmos a compilação, e em seguida abra-o com o seu editor de texto preferido, no nosso exemplo usamos o velho e bom SciTE que já vem na instalação do Ruby 1.8.x, e digitamos o seguinte conteúdo.

```
<?xml version="1.0" encoding="utf-8"?>

<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    backgroundGradientColors="[#ffffff, #c0c0c0]"
    width="100%"
    height="100%"
    fontSize="11" cornerRadius="2" >

    <mx:Script>
    <![CDATA[

        import mx.rpc.events.ResultEvent;
        import mx.controls.Alert;

        private function myHelloWord(msg:String):void {
            Alert.show(msg);
        }

    ]]>
</mx:Script>

<mx:TitleWindow width="450" height="110"
    layout="absolute"
    title="Instant Flex on Rails by e-Genial | Simple Hello Word"
    showCloseButton="true"
    cornerRadius="10"
    fontSize="11"
    verticalAlign="middle">

    <mx:FormItem label="Breve texto" fontSize="11" y="10">
    <mx:TextInput name="texto" id="texto" text="" width="144" fontSize="11"/>
    <mx:Button label="Ok" click="myHelloWord(texto.text)" fontSize="12" x="350" y="20"/>
    </mx:FormItem>

</mx:TitleWindow>
```

Bom a estrutura de formatação e linguagem mxml é bem parecida também com HTML facilitando a escrita e interpretação humana. Em primeiro momento declaramos no cabeçalho que o tipo de documento é XML e que usamos codificação UTF-8:

```
<?xml version="1.0" encoding="utf-8"?>
```

Como em qualquer documento xml correto?

Já a segunda linha inicia o corpo da nossa aplicação:

7 Iniciando o corpo da aplicação

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
    layout="vertical"
    backgroundGradientColors="#ffffff, #c0c0c0"
    width="100%"
    height="100%"
    fontSize="11" cornerRadius="2" >
```

Isso já é um padrão de documentos/fontes de um projeto Flex, pois a partir desse início <mx:Application que nossa aplicação começa de verdade, acrescentamos também alguns parâmetros que define as cores onde criamos um degrade até passarmos por bordas arredondadas em cornerRaius.

8 ActionScript

Vamos agora abrir as tags para adicionar um script AS 3.0 ou mais conhecida como ActionScript, onde criamos uma pequena "função" por onde passamos o texto digitado no campo que por fim mostra o alert. Gostaria de lembrar que a linguagem ActionScript é muito parecida com Javascript, totalmente orientada a objetos e de fácil escrita, a única diferença até agora comparada é somente alguns métodos adicionados a estrutura da linguagem para pacotes do próprio flash/flex para tratar seus componentes internos.

Sempre que você for adicionar um script dentro de um arquivo com estrutura xml você deve separar o conteúdo usando o Character Data:

```
<![CDATA[ ..... ]]>
```

<!-- Inicia a tag onde adicionamos código ActionScript ao projeto -->

<mx:Script>

<! – Aqui finalizamos com o fechamento da tag -->

</mx:Script>

<mx:Script>

<![CDATA[

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

```
    private function myHelloWord(msg:String):void {
        Alert.show(msg);
    }
```

]]

</mx:Script>

Bom aqui creio que não deva ser difícil de entender, na verdade primeiramente importamos os pacotes **mx.rpc.events.ResultEvent;** e o **mx.controls.Alert;** onde o primeiro é para manipularmos os eventos futuros da nossa aplicação e o segundo para usarmos o Alert.

Já em:

```
    private function myHelloWord(msg:String):void {
        Alert.show(msg);
    }
```

Criamos uma função chamada myHelloWord que receberá como parâmetro texto digitado no campo, em seguida mostramos o Alert com o conteúdo do texto.

9 Criando uma window

Bom como estamos trabalhando com uma aplicação rica porque então não criar uma pequena janela para acomodar o campo e o botão?

```
<mx:TitleWindow width="450" height="110"  
  
    layout="absolute"  
  
    title="Instant Flex on Rails by e-Genial | Simple Hello Word"  
  
    showCloseButton="true"  
  
    cornerRadius="10"  
  
    fontSize="11"  
  
    verticalAlign="middle">
```

Fácil iniciar e criar uma window não? Isso faz a grande diferença do flex, pois ele tem uma série de componentes visuais e não visual pronto para uso, basta dar uma breve olhada na documentação e ver o poder desse cara.

10 Criando os elementos campos e botão

```
<mx:FormItem label="Breve texto" fontSize="11" y="10">  
    <mx:TextInput name="texto"  
        id="texto"  
        text=""  
        width="144"  
        fontSize="11"/>  
  
    <mx:Button label="Ok"  
        click="myHelloWord(texto.text)"  
        fontSize="12"  
        x="350"  
        y="20"/>  
</mx:FormItem>
```

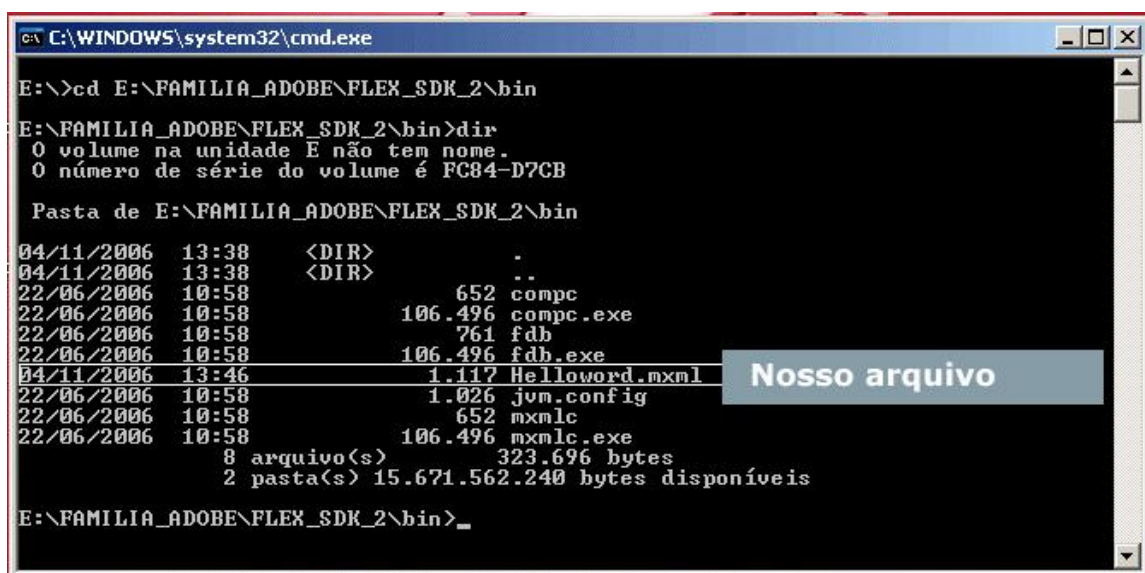
Por fim montamos uma pequena estrutura dentro de FormItem com um TextInput e um Button, para o TextInput adicionamos um nome bem sugestivo "texto", já no Button adicionamos um evento "click" que chamará nossa função myHelloWord passando como parâmetro o que foi digitado no campo texto, para recuperar o valor digitado basta somente fazer: texto.text em qualquer ponto de nosso código, então temos elemento.text.

Aqui finalizamos nossa window e nosso projeto:

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

11 Compilando e testando

Tranquilo até aqui? Então vamos compilar o mxml, no menu iniciar -> executar -> digite cmd, irá abrir o prompt do ms-dos, em seguida digite cd diretorio_onde_você_extraiu o flex assim:



```
C:\WINDOWS\system32\cmd.exe  
E:\>cd E:\FAMILIA_ADOBE\FLEX_SDK_2\bin  
E:\FAMILIA_ADOBE\FLEX_SDK_2\bin>dir  
O volume na unidade E não tem nome.  
O número de série do volume é FC84-D7CB  
  
Pasta de E:\FAMILIA_ADOBE\FLEX_SDK_2\bin  
04/11/2006 13:38 <DIR> .  
04/11/2006 13:38 <DIR> ..  
22/06/2006 10:58 652 compc  
22/06/2006 10:58 106.496 compc.exe  
22/06/2006 10:58 761 fdb  
22/06/2006 10:58 106.496 fdb.exe  
04/11/2006 13:46 1.117 Helloworld.mxml  
22/06/2006 10:58 1.026 jum.config  
22/06/2006 10:58 652 mxmlc  
22/06/2006 10:58 106.496 mxmlc.exe  
8 arquivo(s) 323.696 bytes  
2 pasta(s) 15.671.562.240 bytes disponíveis  
E:\FAMILIA_ADOBE\FLEX_SDK_2\bin>_
```

Nosso arquivo

Mais antes gostaria de dizer que o diretório do Flex 2 SDK vem com os seguintes arquivos e estrutura:

Flex Debugger: **fdb.exe** -> Debugger por linha de comando.

Flex Compiler: **mxmlc.exe** -> compilador por linha de comando onde você gera seu SWF final da aplicação.

Flex Compiler Component: **compc.exe** -> Compilador onde você pode escrever apenas Actionscript 3 e ele compila o seu .as

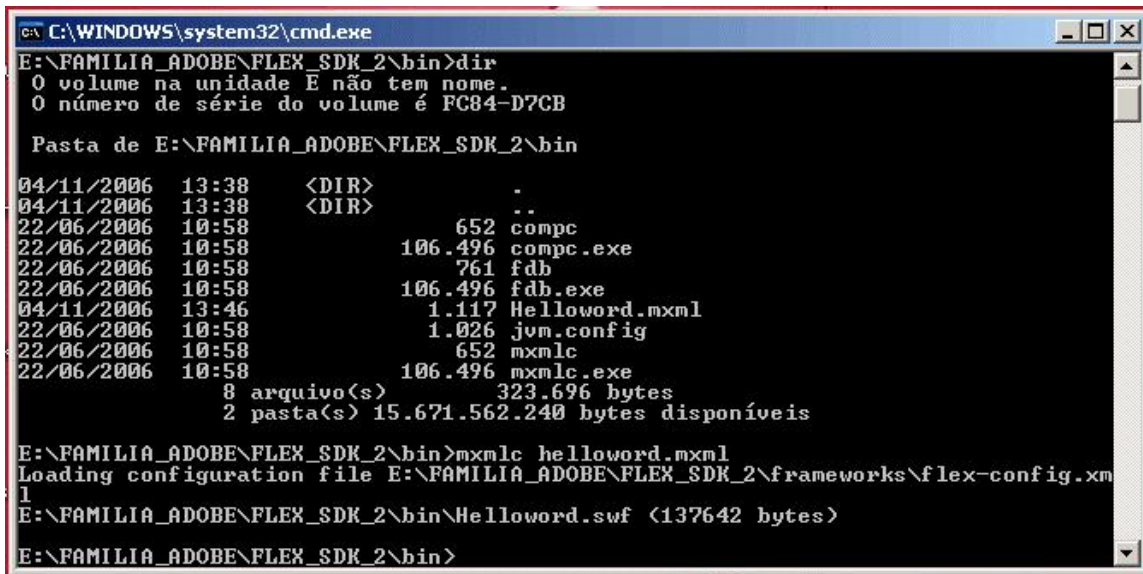
Flash Player Install: Contém versões do Flash Player 9 para instalação.

Lib: Diretório onde contém bibliotecas usadas pelos compiladores para Interpretar seus códigos em MXML e Actionscript 3

Framework: Diretório onde contém o código fonte de todos os componentes

Usados em sua aplicação

Ainda dentro do diretório bin, digite `mxmhc helloworld.mxml` o compilador irá compilar o nosso arquivo e criar um arquivo chamado `Helloworld.swf` como na imagem abaixo:



```
C:\WINDOWS\system32\cmd.exe
E:\FAMILIA_ADOBE\FLEX_SDK_2\bin>dir
O volume na unidade E não tem nome.
O número de série do volume é FC84-D7CB

Pasta de E:\FAMILIA_ADOBE\FLEX_SDK_2\bin

04/11/2006 13:38 <DIR>      .
04/11/2006 13:38 <DIR>      ..
22/06/2006 10:58             652 compc
22/06/2006 10:58          106.496 compc.exe
22/06/2006 10:58             761 fdb
22/06/2006 10:58          106.496 fdb.exe
04/11/2006 13:46             1.117 Helloworld.mxml
22/06/2006 10:58             1.026 jvm.config
22/06/2006 10:58             652 mxmhc
22/06/2006 10:58          106.496 mxmhc.exe
      8 arquivo(s)          323.696 bytes
      2 pasta(s) 15.671.562.240 bytes disponíveis

E:\FAMILIA_ADOBE\FLEX_SDK_2\bin>mxmhc helloworld.mxml
Loading configuration file E:\FAMILIA_ADOBE\FLEX_SDK_2\frameworks\flex-config.xml
1
E:\FAMILIA_ADOBE\FLEX_SDK_2\bin\Helloworld.swf (137642 bytes)

E:\FAMILIA_ADOBE\FLEX_SDK_2\bin>
```

Se tiver algum erro de sintaxe ou algum outro erro o compilador vai te mostrar a linha do erro e quantos erros foram encontrados, mais caso tenha ocorrido tudo certo você deverá ver a tela acima indicando que a compilação foi realizada com sucesso.

Bom se tudo ocorreu 100% e você chegou aqui é hora de testar.

Abra o arquivo `Helloworld.swf` dentro de qualquer browser ou com o botão direito em cima do arquivo abra com o Flash Player, em qualquer uma das opções você irá ver a tela abaixo, ai basta testar digitando algo no campo e clicar no botão você deverá ver o alert.

word.swf



Caso você não veja nada, é porque a versão do teu flash player pode ser antiga, então baixe a última versão que é a 9, em seguida instale e faça novamente o teste.

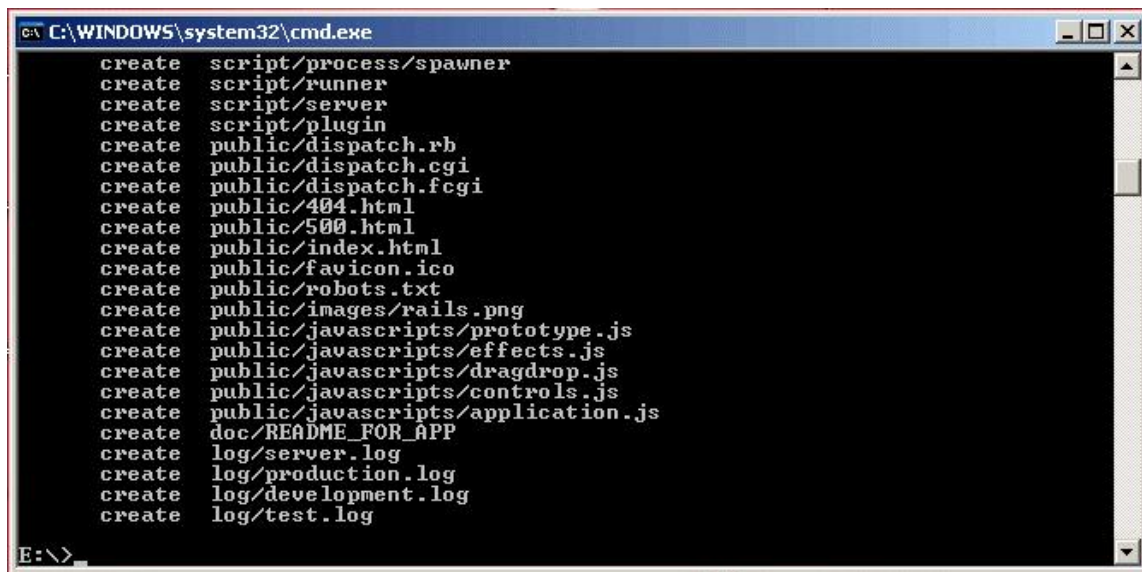
12 Criando uma simples aplicação Rails

Agora vamos criar nossa aplicação rails que vai realizar a comunicação via SOAP com a camada view criada e gerada pelo flex.

Ainda com o cmd/terminal aberto vamos criar o esqueleto da nossa aplicação digitando o seguinte comando:

rails flexonrails

A última saída deverá ser:



```
C:\WINDOWS\system32\cmd.exe
create script/process/spawner
create script/runner
create script/server
create script/plugin
create public/dispatch.rb
create public/dispatch.cgi
create public/dispatch.fcgi
create public/404.html
create public/500.html
create public/index.html
create public/favicon.ico
create public/robots.txt
create public/images/rails.png
create public/javascripts/prototype.js
create public/javascripts/effects.js
create public/javascripts/dragdrop.js
create public/javascripts/controls.js
create public/javascripts/application.js
create doc/README_FOR_APP
create log/server.log
create log/production.log
create log/development.log
create log/test.log
E:\>
```

O próximo passo é criar um controller que vai conter as actions que a aplicação vai precisar, algo como gerar uma saída xml, receber parâmetros e gravar na sessão, apagar um elemento dentro de um array da sessão ou apagar todos os registros da sessão algo rápido e direto. Então digite o seguinte comando:

cd flexonrails

ruby script/generate controller acao

Em seguida você poderá já editar o arquivo **acao_controller.rb** que esta dentro de flexonrails\app\controllers e adicionar o seguinte conteúdo.

```
def inserir
  if @session[:valores] == nil
    @session[:valores] = []
  end
  @session[:valores].push(:parametro=>params[:valor])
  if @session != nil
    render :xml =>@session[:valores].to_xml
  else
    render(:nothing => true)
  end
end

def lista
  if @session[:valores] != nil
    render :xml =>@session[:valores].to_xml
  else
    render(:nothing => true)
  end
end

def apagar
  @session[:valores] = []
  render :text=>"ok"
end

def apaga_item
  if params[:valor] != nil && params[:valor] != ""
    @session[:valores].delete(:parametro=>params[:valor])
    render :text=>"ok"
  else
    render :text=>"false"
  end
end
```

Extremamente fácil Ruby on Rails não? ;-)

O método **inserir** é responsável por receber um parâmetro chamado **:valor** que grava o conteúdo do parâmetro dentro de cada posição do Array **@session[:valores]** persistindo assim seus valores na sessão, e, se a sessão for realmente diferente de nula ele renderiza um xml na saída, o rails possui um método chamado `to_xml` pra fazer isso automaticamente, onde ele recebe um array de hashes para gerar os nós e conteúdos de um xml.

O método **lista** é responsável somente por gerar o xml com todo o conteúdo de nossa array que esta na sessão.

Já o método **apagar** limpa o array inteiro, zerando todo seu conteúdo.

E por fim o método **apagar_item** que recebe como parâmetro novamente **:valor** onde comparamos se o que foi passado é realmente diferente de nulo caso seja deletamos o item de dentro do array **@session[:valores]** usando o método **.delete** e em seguida retornamos um ok dizendo que foi deletado.

13 Criando a camada de visualização com Flex

Bom já temos toda a “regra de negócio” criada, agora vamos criar a integração e comunicação da camada view gerada pelo flex com a aplicação Rails.

Agora crie um arquivo chamado **flexonrails.mxml** dentro da pasta bin mesmo somente para adiantarmos a compilação, em seguida abra-o com o seu editor de texto preferido, neste exemplo usamos o velho e bom SciTE que já vem na instalação do Ruby 1.8.x, e digitamos o seguinte conteúdo.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"  
    layout="vertical"  
    backgroundGradientColors=["#ffffff, #c0c0c0]"  
    width="100%"  
    height="100%"  
    creationComplete="retorno.send()"  
    viewSourceURL="srcview/index.html"  
    fontSize="11" cornerRadius="2" >
```

```
<mx:Script>
```

```
<![CDATA[
```

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

```
private function resultadoDelecao(event:ResultEvent):void {  
    var result:Object = event.result;  
    if (result == "ok") {  
        Alert.show("Deletado com sucesso todos \n os registros da sessão !");  
        retorno.send();  
        valor.text="";  
    }  
}
```

```
private function resultadoDelecaoItem(event:ResultEvent):void {  
    var result:Object = event.result;  
    if (result == "ok") {  
        Alert.show("Registro Deletado com sucesso !");  
        retorno.send();  
    }  
}
```

```
]]>
```

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

```
<mx:HTTPService id="enviados"
    url="http://localhost:3000/acao/inserir"
    useProxy="false"
    method="POST"
    result="retorno.send(); valor.text=";
    contentType="application/xml">
    <mx:request xmlns="">
        <valor>{valor.text}</valor>
    </mx:request>
</mx:HTTPService>

<mx:HTTPService
    id="retorno"
    url="http://localhost:3000/acao/lista" />

<mx:HTTPService
    id="deleta"
    url="http://localhost:3000/acao/apagar"
    result="resultadoDelecao(event)" />

<mx:HTTPService id="apagaitem"
    url="http://localhost:3000/acao/apaga_item"
    useProxy="false"
    method="POST"
    result="resultadoDelecaoItem(event)">
    <mx:request xmlns="">
        <valor>{conteudo.selectedItem.parametro}</valor>
    </mx:request>
</mx:HTTPService>
```

```

<mx:Panel width="302" height="152" layout="absolute" title="Mini-Form" fontSize="11">
  <mx:Form width="262" x="10" y="10" height="57" backgroundColor="#ffffff">
    <mx:FormItem label="Breve texto" fontSize="11">
      <mx:TextInput name="valor" id="valor" text="" width="144" fontSize="11"/>
    </mx:FormItem>
  </mx:Form>
  <mx:Button label="Enviar" click="enviados.send()" fontSize="12" x="85" y="87"/>
  <mx:Button label="Limpar sessão" click="deleta.send()" fontSize="11" x="161" y="87"/>
</mx:Panel>

<mx:TitleWindow width="358" height="338"
  layout="absolute"
  title="Flex on Rails persistindo dados na sessão ;-)"
  showCloseButton="true"
  cornerRadius="10"
  fontSize="11"
  verticalAlign="middle">

  <mx:DataGrid width="338" height="297" id="conteudo"
    dataProvider="{retorno.lastResult.records.record}" fontSize="11" x="0"
    click="apagaitem.send();" y="0">

    <mx:columns>
      <mx:DataGridColumn headerText="Parâmetros enviados" dataField="parametro" />
    </mx:columns>
  </mx:DataGrid>

</mx:TitleWindow>

</mx:Application>

```

Chega até parecer complexo não? É, mas não é nada assustador não, muito pelo contrario, grande parte desde código já discutimos no nosso primeiro exemplo, o grande diferencial é que ele possui algumas tags responsáveis pela comunicação com nossa aplicação Rails, veja:

<mx:HTTPService

Esta tag **HTTPService** representa um objeto dentro do pacote mx.rpc.http.mxml, quando você chama o método HTTPService.send() ele envia como pedido um GET para a url solicitada e em seguida é retornado uma resposta desta conexão, esta tag aceita vários parâmetros para ser enviados para a URL solicitada.

Gostaria de ressaltar aqui, que o parâmetro url da tag **HTTPService**

url= ` <http://localhost:3000/acao/insereir> `

foi usado desta maneira com o número da porta porque estamos usando o webrich localmente, mas em produção poderíamos colocar somente

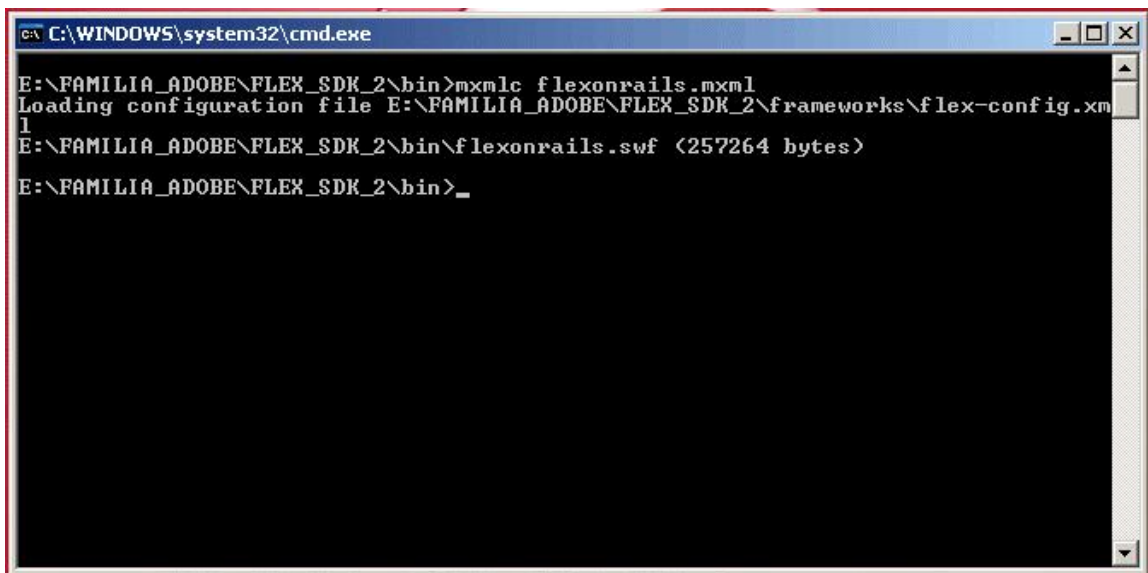
/ação/inserir ou seja o caminho relativo por exemplo, pois em produção não existe a porta 3000 então logo fica transparente.

Criamos também um painel com a tag **<mx:Panel** para alojar nosso pequeno formulário, e por fim criamos uma grid com a tag **<mx:DataGrid** que recebe um dataProvider com o retorno do nosso xml gerado pela action **lista** do controler ação. Os itens são mostrados em cada linha da grid que ao ser selecionada irá disparar o HTTPService identificado por **apagaitem**, então temos no evento click **apagaitem.send()**; que em seguida passa como parâmetro o valor do registro em **{conteudo.selectedItem.parametro}**.

Bom agora vamos compilar o arquivo flexonrails.mxml com o seguinte comando:

mxmhc flexonrails.xml

A saída deverá ser a da imagem abaixo:

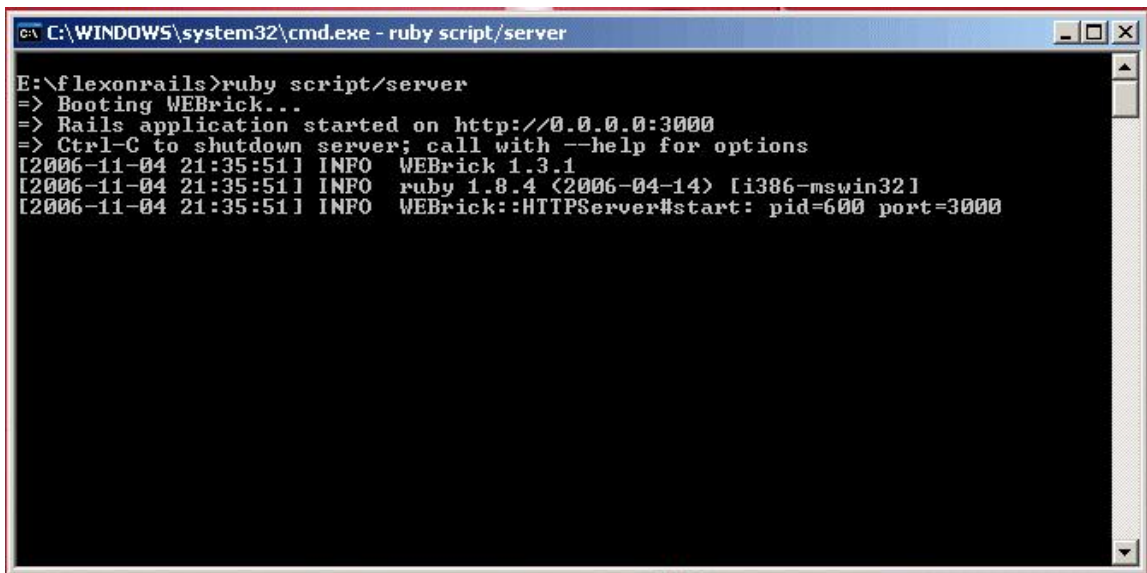


```
C:\WINDOWS\system32\cmd.exe
E:\FAMILIA_ADOBE\FLEX_SDK_2\bin>mxmhc flexonrails.mxml
Loading configuration file E:\FAMILIA_ADOBE\FLEX_SDK_2\frameworks\flex-config.xml
1
E:\FAMILIA_ADOBE\FLEX_SDK_2\bin\flexonrails.swf (257264 bytes)
E:\FAMILIA_ADOBE\FLEX_SDK_2\bin>_
```

Se você chegou até aqui é por tudo ocorreu 100%, agora vamos iniciar nossa aplicação rails com o seguinte comando:

ruby script/server

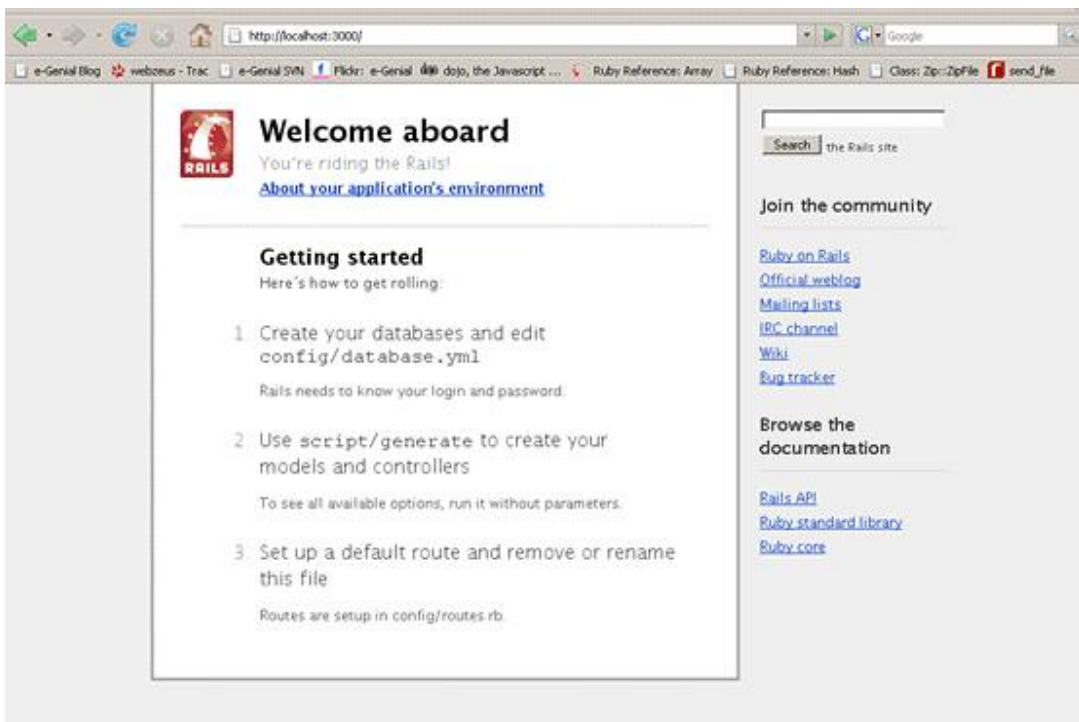
A saída deverá ser conforme a imagem abaixo



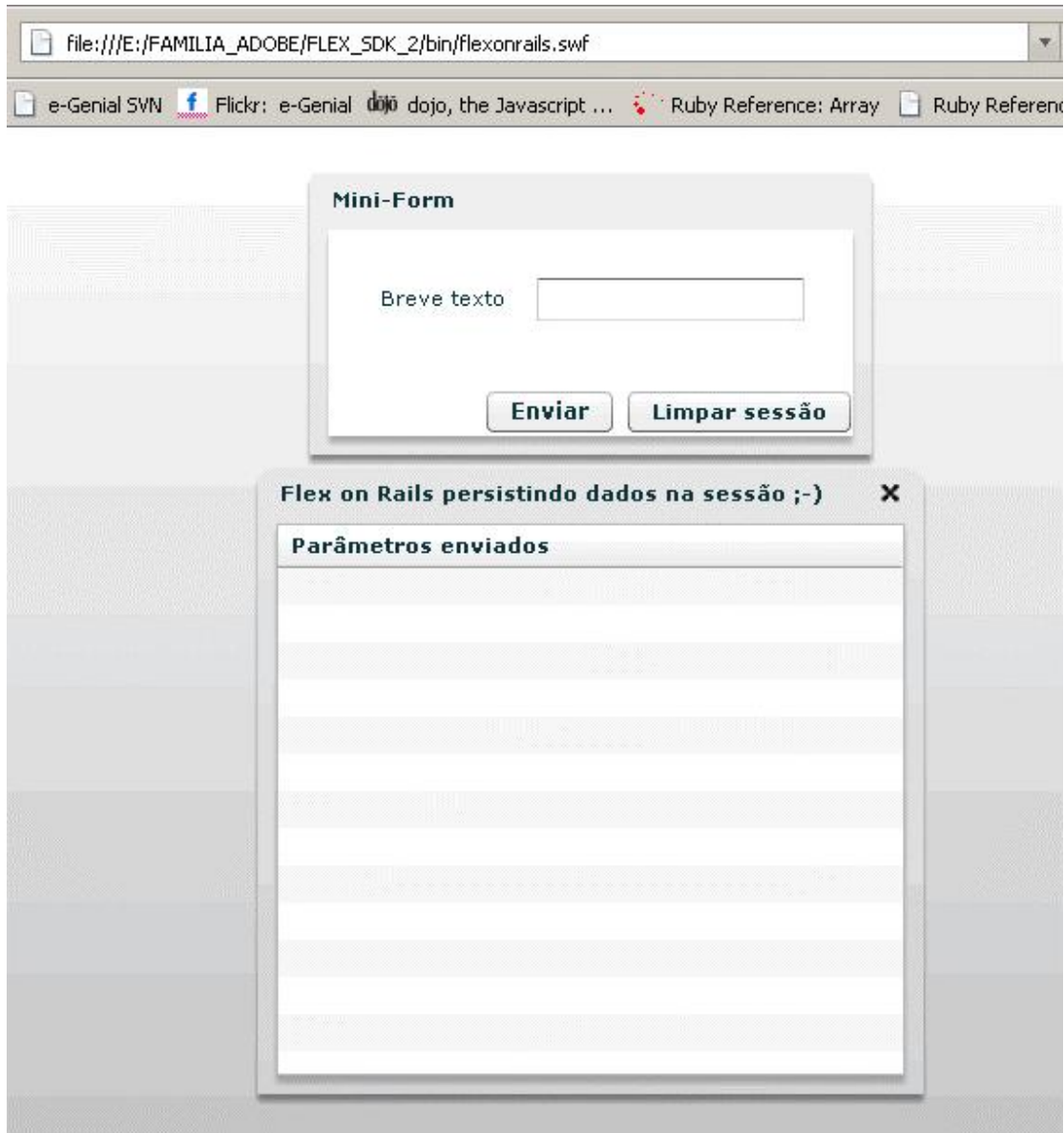
```
C:\WINDOWS\system32\cmd.exe - ruby script/server

E:\flexonrails>ruby script/server
=> Booting WEBrick...
=> Rails application started on http://0.0.0.0:3000
=> Ctrl-C to shutdown server; call with --help for options
[2006-11-04 21:35:51] INFO WEBrick 1.3.1
[2006-11-04 21:35:51] INFO ruby 1.8.4 (2006-04-14) [i386-mswin32]
[2006-11-04 21:35:51] INFO WEBrick::HTTPServer#start: pid=600 port=3000
```

Acesse a seguinte url pelo browser para testar se foi realmente levantada sua aplicação <http://localhost:3000/> você deverá ver a seguinte tela



Tranquilo até aqui, agora é hora de testar se esta tudo certo abra o arquivo flexonrails.swf que foi gerado pelo compilador mxmhc, você verá.



Agora é só testar. Adicione um breve texto no campo, em seguida clique no botão enviar, e na seqüência verá o que digitou em uma linha na grid, depois clique na linha da grid para deletar o item adicionado ou clique diretamente no botão Limpar sessão para limpar tudo o que você fez.

14 Como aprender mais?

Gostaria aqui de colocar aqui quatro pontos básicos que pode dar uma breve ajuda inicial para aprender e conhecer mais sobre Flex e a comunicação com Rails.

- 1 - Você agora neste ponto poderá acompanhar os logs gerado pela aplicação rails para entender como esta funcionando todos os processos de envio e recebimento do xml gerado.
- 2 - Acesse diretamente a seguinte url <http://localhost:3000/acao/lista> e veja como estão sendo listados os dados que estão na sessão.
- 3 - Crie mais um método dentro do controler ação para atualizar os elementos selecionados na grid em vez de deletar cada item.
- 4 - Dentro do diretório que você descompactou o Flex 2 SDK existe um diretório chamado samples, dentro dele existe um arquivo chamado build-samples.bat para usuários do windows ou build-samples.sh para usuários linux esse arquivo contem uma macro para compilar todos os exemplos contido nesta pasta, depois de compilado você terá uma grande documentação e exemplos gerados para entender mais sobre componentes do flex.

Qualquer dúvida, peço a gentileza que me envie um e-mail que terei grande satisfação em ajudar.

Grande abraço e até a próxima...