

Développez des applications RIA avec Richfaces

par [Lotfi Mellouk](#)

Date de publication : 30/12/08

Dernière mise à jour :

Cet article fait l'objet d'une introduction au développement d'applications Web riches avec JSF et Ajax/Richfaces.

I - Avant propos.....	4
I-A - Présentation.....	4
I-B - Pré-requis.....	4
II - Ajax4JSF.....	4
II-A - Introduction.....	4
II-B - Comment ça marche ?.....	4
III - Installation.....	5
III-A - Compatibilité.....	5
IV - Exemple d'application Richfaces.....	6
IV-A - Création du projet web.....	6
IV-B - Configuration des dépendances avec Maven.....	6
IV-C - Configuration du filtre Richfaces.....	7
IV-D - Utilisation de Richfaces avec facelets.....	8
IV-E - Première page Richfaces.....	8
V - Les principaux composants a4j.....	11
V-A - <a4j:support />.....	11
V-B - <a4j:commandButton />.....	11
V-C - <a4j:commandLink/>.....	12
V-D - <a4j:region/>.....	12
V-E - <a4j:keepAlive/>.....	12
V-F - <a4j:poll/>.....	12
V-G - <a4j:outputPanel/>.....	13
V-H - <a4j:actionparam/>.....	13
VI - Les principaux composants de Richfaces.....	13
VI-A - <rich:tabPanel/>.....	13
VI-B - <rich:panel/>.....	14
VI-C - <rich:panelBar/>.....	14
VI-D - <rich:simpleTogglePanel />.....	15
VI-E - <rich:modalPanel/>.....	15
VI-F - <rich:dataTable/>.....	16
VI-G - <rich:DataTableScroller/>.....	18
VI-H - <rich:contextMenu />.....	18
VI-I - <rich:jQuery />.....	21
VI-J - <rich:ajaxValidator/>.....	21
VII - Améliorer la performance des applications Richfaces.....	22
VII-A - Contrôler la file d'attente des requêtes Ajax.....	22
VII-B - Contrôler l'exécution des phases JSF.....	22
VIII - Richfaces skins.....	23
VIII-A - Présentation.....	23
VIII-B - Utiliser un skin.....	23
VIII-C - Personnaliser un skin.....	24
VIII-D - Format XCSS.....	25
VIII-E - Utiliser les skins Richfaces pour les composants HTML.....	26
IX - Astuces et questions fréquentes.....	27
IX-A - [rich:TabPanel] comment définir l'onglet courant :.....	27
IX-B - [rich:TabPanel] Comment réaliser une navigation en cliquant sur un onglet.....	27
IX-C - [rich:TabPanel] Comment définir son propre style des onglets.....	28
IX-D - [rich :TabPanel] Comment créer dynamiquement TabPanel.....	28
IX-E - [rich:dataTable] : comment afficher un modalPanel à la fin de l'exécution d'une action sur une ligne.....	30
IX-F - [rich :dataTable] : comment définir un style différent pour deux lignes successives.....	30
IX-G - [rich :dataTable] : comment filtrer les données dans le managed Bean.....	30
IX-H - [rich:dataTable] : comment réduire la taille du champ texte généré par filterBy.....	30
IX-I - [rich :dataTable] : comment cacher/afficher dynamiquement une colonne.....	31
IX-J - [rich :dataTable] [rich :dataScroller] : comment sélectionner la page courante avec un selectOneMenu :.....	31
IX-K - [rich :dataTable] [rich:dataScroller] : comment mettre à jour le dataScroller après l'utilisation d'un filtre:.....	32
IX-L - [rich:modalPanel] comment ajouter un scroller à la modalPanel.....	32

IX-M - [rich:modalPanel] Comment afficher un modalPanel au chargement d'une page.....	32
IX-N - [rich:modalPanel] comment afficher les messages d'erreurs de validation dans un modalPanel.....	32
IX-O - Comment gérer l'expiration de session avec Richfaces.....	33
IX-P - [rich:comboBox] comment sélectionner un élément par son id ?.....	34
IX-Q - [rich:suggestionBox] comment récupérer l'id d'un élément sélectionné?.....	34
X - Remerciements.....	34

I - Avant propos

I-A - Présentation

Ce tutorial est une introduction au développement d'applications riches avec JSF /Richfaces. Richfaces est une librairie de composants JSF pour le développement d'applications web riches (RIA) Rich Internet Application avec Ajax, elle est le résultat de la fusion de deux projets développés par exadel qui sont:

- **Ajax4jsf** :
Ajax4jsf est né dans la fondation sourceforge.net en 2005 sous le nom de Telamon, son concepteur a intégré ensuite la société Exadel qui a commercialisé le projet dans une première période avant de le rendre open source sur Java.net.
- **Richfaces** :
C'est une librairie commerciale de composants JSF fournit par Exadel.

Ces deux projets sont passés dans le giron de JBoss en septembre 2007 pour former un projet open source nommé JBoss Richfaces.

I-B - Pré-requis

- La maîtrise de la technologie JSF, pour les débutants. Voir tutoriels: [Introduction au framework JSF](#) de Mickael BARON, [Gagner en productivité avec Java Server Faces](#) de Xavier Paradon, et [JSF et Facelets](#) de Jawher Moussa.
- Compréhension du fonctionnement Ajax. [Voir tutoriaux Ajax](#).

II - Ajax4JSF

II-A - Introduction

Ajax4jsf est né du besoin d'associer la technologie Ajax à celle de JSF, il comporte un ensemble de composants permettant d'ajouter des fonctionnalités Ajax avancées aux composants standards JSF sans manipulation du code JavaScript.

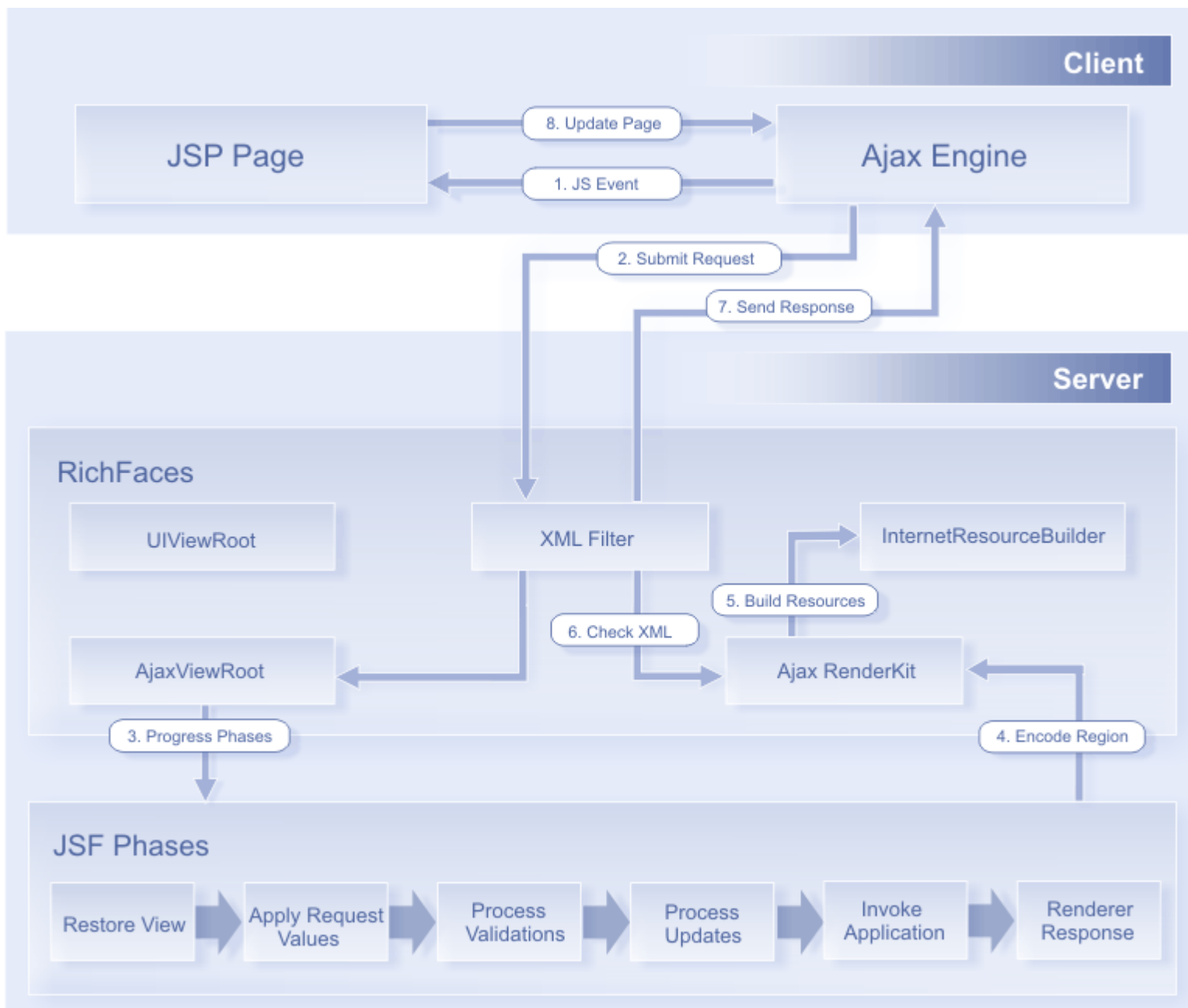
II-B - Comment ça marche ?

Ajax4jsf est basé sur un filtre qui ajoute des fonctions JavaScript ainsi que les dépendances de **XmlHttpRequest** aux composants JSF. Un moteur Ajax est responsable de l'envoi de la requête cliente au filtre.

Lorsque le client envoie une requête Ajax, le moteur Ajax transforme la requête et l'envoie au filtre **ajax4jsf**, ce dernier convertit les données au format XML et transfère la requête à la servlet Faces Servlet, ainsi la requête suit le processus normal d'une requête JSF et passe par les six phases : **Restore View**, **Apply Request Values**, **Process Validations**, **Update Model Values**, **Invoke Application**, **Render Response**.

Le Framework ajax4jsf gère l'arbre de composants tout au long de la requête et la réponse Ajax. De plus, la présence du moteur Ajax dans le client permet la mise à jour de parties souhaitées de la page Web, ce qui fait de Ajax4JSF un framework très flexible qui offre au développeur la possibilité de choisir les composants à envoyer au serveur et les valeurs à réactualiser à la fin de la réponse.

L'image ci-dessous présente le cycle de vie d'une requête Ajax4jsf



III - Installation

La dernière version publiée de Richfaces est disponible en téléchargement sur le site de **JBoss**. Les trois jars suivants sont nécessaires à son utilisation:

- Richfaces-api
- Richfaces-ui
- Richfaces-impl

III-A - Compatibilité

A partir de la version 3.2.0, Richfaces supporte uniquement :

- JSF 1.2 (JSF RI 1.2 et Myfaces 1.2)
- Java SE 5.0

Toutefois, l'équipe Richfaces maintient la mise à jour des versions 3.1.x, compatibles avec JSF 1.1 et Jdk 1.4.

IV - Exemple d'application Richfaces

Dans ce chapitre nous allons réaliser une première application avec Richfaces, JSF 1.2, Facelets avec Eclipse et Maven sur le serveur d'applications tomcat 5.5. Pour utiliser tomcat 6, pensez à enlever les api EL (el-*.jar) qui sont disponibles par défaut dans tomcat6.

IV-A - Création du projet web

Nous commençons par créer un projet web avec maven et les fichiers de configuration d'eclipse

Créer un projet web avec maven

```
mvn archetype:create -DgroupId=tutorial -DartifactId=Richfaces -DarchetypeArtifactId=maven-archetype-webapp
```

Ensuite dans le dossier Richfaces

généraliser les fichiers de configuration d'Eclipse

```
mvn eclipse:eclipse -Dwtpversion=1.5
```

et on importe le projet sous eclipse.

IV-B - Configuration des dépendances avec Maven

Les dépendances dont nous avons besoin sont définies dans fichier pom.xml.

maven dependencies

```
<dependencies>
  <dependency>
    <groupId>javax.faces</groupId>
    <artifactId>jsf-api</artifactId>
    <version>1.2_09</version>
  </dependency>
  <dependency>
    <groupId>javax.faces</groupId>
    <artifactId>jsf-impl</artifactId>
    <version>1.2_09</version>
    <scope>runtime</scope>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>jsp-api</artifactId>
    <version>2.1</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.1.2</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

maven dependencies

```
<dependency>
  <groupId>javax.el</groupId>
  <artifactId>el-api</artifactId>
  <version>1.2</version>
</dependency>
<dependency>
  <groupId>com.sun</groupId>
  <artifactId>el-ri</artifactId>
  <version>1.2</version>
</dependency>
<dependency>
  <groupId>com.sun.facelets</groupId>
  <artifactId>jsf-facelets</artifactId>
  <version>1.1.14</version>
</dependency>
<dependency>
  <groupId>org.richfaces.framework</groupId>
  <artifactId>richfaces-impl</artifactId>
  <version>3.2.2.GA</version>
</dependency>
<dependency>
  <groupId>org.richfaces.framework</groupId>
  <artifactId>richfaces-api</artifactId>
  <version>3.2.2.GA</version>
</dependency>
<dependency>
  <groupId>org.richfaces.ui</groupId>
  <artifactId>richfaces-ui</artifactId>
  <version>3.2.2.GA</version>
</dependency>
<dependency>
  <groupId>commons-logging</groupId>
  <artifactId>commons-logging</artifactId>
  <version>1.0.4</version>
</dependency>
</dependencies>
```

Ensuite il faut ajouter la déclaration du repository JBoss qui contient l'ensemble des projets Richfaces

JBoss repository

```
<repository>
  <id>jboss</id>
  <name>jboss repository</name>
  <url>http://repository.jboss.com/maven2/</url>
</repository>
```

Et le repository pour Facelets et el :

dev.java.net repository

```
<repository>
  <id>dev.jav</id>
  <name>nonav</name>
  <url>https://maven-repository.dev.java.net/repository/</url>
  <releases>
    <enabled>true</enabled>
  </releases>
</repository>
```

IV-C - Configuration du filtre Richfaces

On ajoute le filtre Ajax4jsf qui parse le contenu Html en XML pour les requêtes Ajax.

définition du filtre Ajax4jsf

```
<filter>
  <display-name>RichFaces Filter</display-name>
  <filter-name>richfaces</filter-name>
  <filter-class>org.ajax4jsf.Filter</filter-class>
</filter>
<filter-mapping>
  <filter-name>richfaces</filter-name>
  <servlet-name>Faces Servlet</servlet-name>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
```

IV-D - Utilisation de Richfaces avec facelets

Pour utiliser Richfaces avec facelets, il faut définir le viewHandler de Richfaces, et remplacer le suffixe par défaut des pages JSF en xhtml. Dans le fichier de configuration web.xml :

configuration Facelets

```
<context-param>
  <param-name>org.ajax4jsf.VIEW_HANDLERS</param-name>
  <param-value>com.sun.facelets.FaceletViewHandler</param-value>
</context-param>
<context-param>
  <param-name>javax.faces.DEFAULT_SUFFIX</param-name>
  <param-value>.xhtml</param-value>
</context-param>
```

IV-E - Première page Richfaces

Nous sommes maintenant en mesure de développer une page JSF avec des fonctionnalités Ajax. Nous allons créer notre première page index.xhtml, on la déclare comme page d'accueil de notre application.



Pour déclarer une page JSF en tant que page d'accueil consultez la [FAQ JSF](#)

Voici la page index.xhtml :

index.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:a4j="http://richfaces.org/a4j"
  xmlns:rich="http://richfaces.org/rich">
  <ui:composition template="./WEB-INF/templates/template.xhtml">

    <ui:define name="title">Richfaces test</ui:define>
    <ui:define name="body">

      <rich:panel style="border:0;width:60%;text-align:center">
        <h:form>

          <rich:panel>
            <f:facet name="header">
              <h:outputText value=" Exemple a4j" />
            </f:facet>
            <h:selectOneMenu value="#{person.name}">
```


index.xhtml

```
<f:selectItem itemLabel="Pierre" itemValue="Pierre" />
<f:selectItem itemLabel="Paul" itemValue="Paul" />
<f:selectItem itemLabel="Jacques" itemValue="Jacques" />
<a4j:support event="onclick" reRender="text">
</a4j:support>
</h:selectOneMenu>

<h:outputText value="Selected name: #{person.name}" id="text"
style="font-weight:bold;" />
</rich:panel>
</h:form>
</rich:panel>
</ui:define>
</ui:composition>
</html>
```

Le template Facelets contient le cadre global de la page, le cadre du bandeau et le cadre pour le contenu, si vous n'êtes pas habitué avec les Facelets, vous pouvez enlever le code relatif au template :

index.xhtml sans Facelets

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:a4j="http://richfaces.org/a4j"
xmlns:rich="http://richfaces.org/rich">
<head>
<title>Richfaces tutorial</title>
</head>
<body>
<rich:panel style="border:0;width:60%;text-align:center">
<h:form>
<rich:panel>
<f:facet name="header">
<h:outputText value="Exemple a4j" />
</f:facet>
<h:selectOneMenu value="#{person.name}">
<f:selectItem itemLabel="Pierre" itemValue="Pierre" />
<f:selectItem itemLabel="Paul" itemValue="Paul" />
<f:selectItem itemLabel="Jacques" itemValue="Jacques" />
<a4j:support event="onclick" reRender="text">
</a4j:support>
</h:selectOneMenu>
<h:outputText value="Selected name: #{person.name}" id="text"
style="font-weight:bold;" />
</rich:panel>
</h:form>
</rich:panel>
</body>
</html>
```

La déclaration des tags Richfaces peut se faire en xml :

déclaration des tags Richfaces

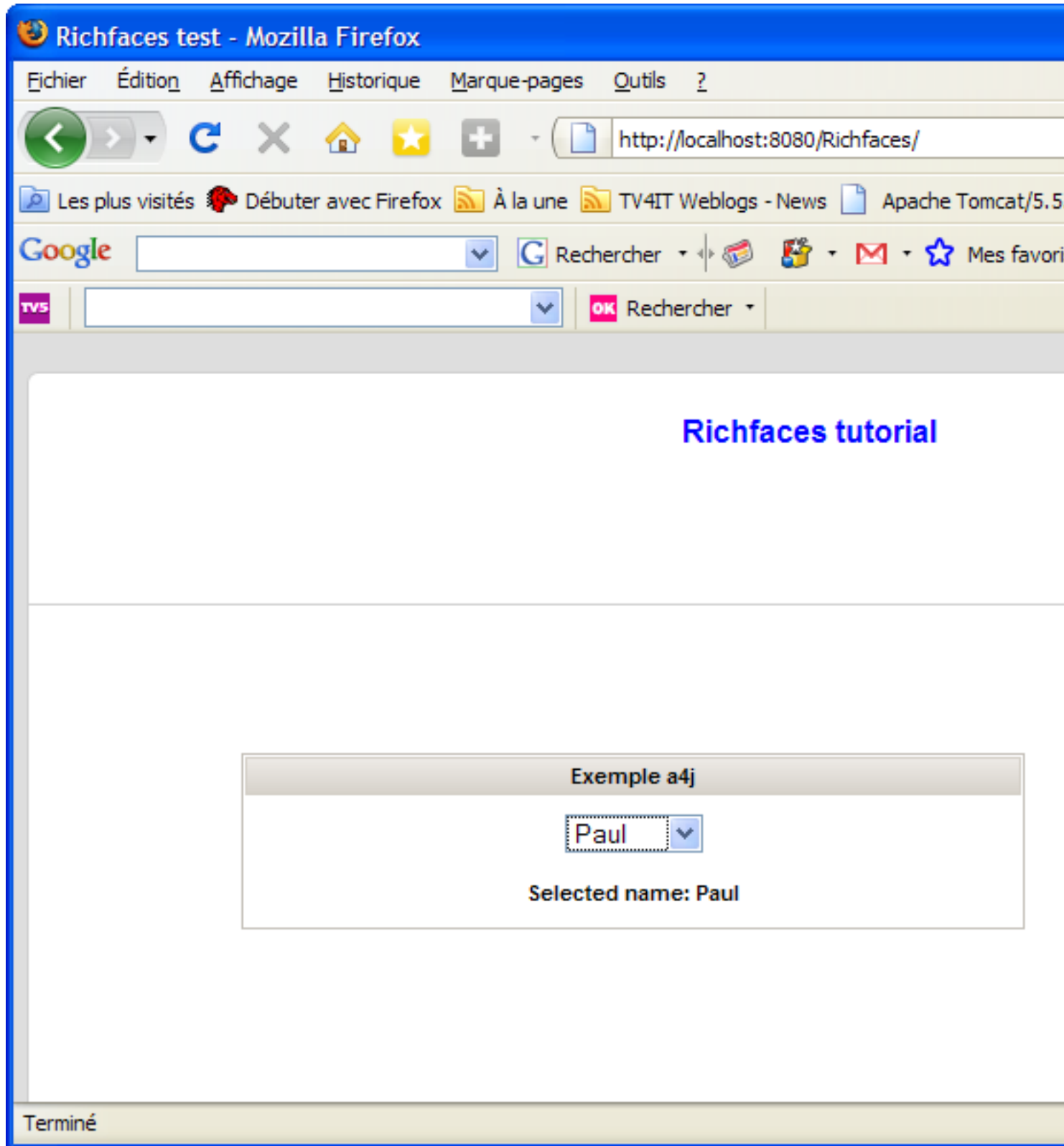
```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:a4j="http://richfaces.org/a4j"
xmlns:rich="http://richfaces.org/rich"/>
```

Et en JSP:

```
<%@ taglib uri="http://richfaces.org/a4j" prefix="a4j"%>
```

```
<%@ taglib uri="http://richfaces.org/rich" prefix="rich"%>
```

le résultat de la page index.xhtml:



V - Les principaux composants a4j

Nous présenterons quelques composants Ajax4jsf, pour trouver la liste exhaustive des composants, veuillez vous référer au site **démo de Richfaces**

V-A - <a4j:support />

Exemple a4j:support

```
<h:commandButton value="Click">
  <a4j:support event="onClick" action="#{bean.monAction}"
    actionListener="#{bean.monActionListener}" reRender="text,panel"></a4j:support>
</h:commandButton>
```

Le plus connu et le plus utile des composants Ajax4jsf, il permet d'ajouter le support Ajax aux composants JSF de base, il s'applique au composant parent, parmi ces attributs :

- **event** : l'évènement javascript qui lance la requête Ajax
- **reRender** : contient les ids des composants à mettre à jours lors du retour de la réponse.
- **actionListener** : binding de la méthode qui prend un ActionEvent en paramètre et retourne un type void
- **action** : binding de la méthode qui invoque l'action de l'application

Les composants a4j ainsi que Richfaces disposent d'attributs communs, parmi ces attributs on peut citer:

- **reRender**: idem que aj4:support
- **ajaxSingle**: booléen pour limiter le passage par les phases **decode**, **validation/conversion**, **apply values**, seulement au composant qui envoie la requête.
- **immediate**: c'est le même principe qu'un composant JSF de base, utile pour éviter les phases validation/conversion
- **bypassUpdates**: permet d'éviter les phases **Update Model** et **Invoke Application** pour passer directement à phase render response, peut être utile pour réduire le temps de réponse lors de la validation des composants
- **ignoreDupResponses**: permet d'ignorer la mise à jour du client pour une réponse dont il existe une autre réponse identique plus récente pour la même requête.
- **limitToList**: si la valeur est *true*, la mise à jour des composants coté client se limite à la liste dans l'attribut reRender
- **timeout**: le temps en ms de la durée de vie d'une requête Ajax
- **oncomplete**: code à exécuter coté client à la fin de la requête
- **process**: liste des ids de composants traités dans les phases 2 à 5 du cycle de vie JSF.
- **data**: permet de récupérer du serveur une nouvelle donnée pendant la requête Ajax, il permet de prendre une propriété d'un bean avec EL, la donnée sera sérialisée dans le format **JSON** pour qu'elle soit disponible coté client.

Le développeur est amené, parfois, à appeler directement du code javascript lors de l'envoi d'une requête Ajax. Ceci est rendu possible à l'aide d'attributs appartenants au composant déclanchant la requête. Ces attributs sont :

- **onsubmit** : juste avant le lancement de la requête.
- **onbeforedomupdate** : avant la mise à jour du DOM.
- **oncomplete** : après la mise à jour du DOM.

V-B - <a4j:commandButton />

Similaire à h:commandButton, mais génère une requête Ajax et fait la mise à jour des composants inclus dans l'attribut reRender.


Attention !

Ce composant n'est pas fait pour réaliser la navigation, à l'instar de tous les composants a4j de contrôle.



Toutefois, si on est contraint de le faire, il existe deux façons de contourner ce cas :

1- utiliser redirect pour la navigation

redirect

```
<navigation-rule>
  <from-view-id>/source.xhtml</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/target.xhtml</to-view-id>
    <redirect />
  </navigation-case>
</navigation-rule>
```

2- l'utiliser à l'intérieur d'un a4j:include

V-C - <a4j:commandLink/>

Similaire à **h:commandLink** avec des fonctionnalités Ajax en plus.

V-D - <a4j:region/>

Il permet de définir la zone de composants qui seront traités coté serveur pour une requêtes Ajax, pendant les phases Decode, update model. Par défaut, la page entière est prise en compte.

```
<a4j:region id="ajaxRegion">
  <!-- autres composants -->
</a4j:region>
```

À chaque requête Ajax , seuls les composants à l'intérieur de la région seront envoyés au serveur.

V-E - <a4j:keepAlive/>

Permet de garder *en vie* un managedBean déclaré dans le scope request entre plusieurs requête Ajax.

```
<a4j:keepAlive beanName="managedBean">
</a4j:keepAlive>
```

V-F - <a4j:poll/>

Permet de rafraîchir périodiquement un composant en envoyant régulièrement des requêtes Ajax dans un intervalle de temps donné.

```
<a4j:poll reRender="panel_to_rerender" interval="1000">
</a4j:poll>
```

V-G - <a4j:outputPanel/>

Ressemble à **h:panelGroup**, il présente les caractéristiques:

- Permet d'agir en Ajax sur des composants non rendus (`rendered="false"`) dans la requête primaire (non Ajax) qui a servi la page. Le composant sera affiché avec les modifications de l'action si la valeur de l'attribut `layout` est égale à `none`: `layout="none"`.
- Permet de rendre automatiquement une zone réactualisable après une requête Ajax sans avoir à le mentionner dans un `reRender`.
- L'utilisation d'affichage conditionnel des composants (`rendered="false/true"`) oblige d'avoir un composant parent qui soit toujours présent dans la request/response comme `a4j:outputPanel`.

V-H - <a4j:actionparam/>

actionparam combine les fonctionnalités de **f:pram** et **f:actionListener**, il permet en plus, d'assigner une valeur à un champ dans un bean. Si la valeur est un objet non reconnu par JSF, il est nécessaire d'ajouter un **convert**.

L'exemple suivant présente une colonne d'une dataTable dans laquelle nous avons ajouté un bouton pour supprimer la ligne. Nous utilisons `actionparam` pour donner la valeur de l'identifiant de l'objet de la ligne courante au `managedBean` :

```
<rich:dataTable value="#{managedBean.users}" var="list">
  <!-- autres colonnes-->
  <rich:column>
    <f:facet name="header">
      <h:outputText styleClass="headerText" value="Action" />
    </f:facet>
    <a4j:commandButton value="Supprimer"
      action="#{managedBean.supprimerUser}" reRender="src">
      <a4j:actionparam name="user_id" value="#{list.userId}"
        assignTo="#{managedBean.selectedUserId}"></a4j:actionparam>
    </a4j:commandButton>
  </rich:column>
</rich:dataTable>
```

VI - Les principaux composants de Richfaces

Les composants Richfaces sont des composants graphiques avancés qui possèdent différentes fonctionnalités Ajax. Ce sont des composants prêts à l'emploi, contrairement aux composants **a4j** qui laissent au développeur le soin de spécifier l'événement Ajax à traiter, la partie de la page à envoyer dans la requête et celle à réactualiser au retour.

VI-A - <rich:tabPanel/>

Principe d'affichage par onglets, supporte trois modes de navigation : **server**, **ajax**, **client**

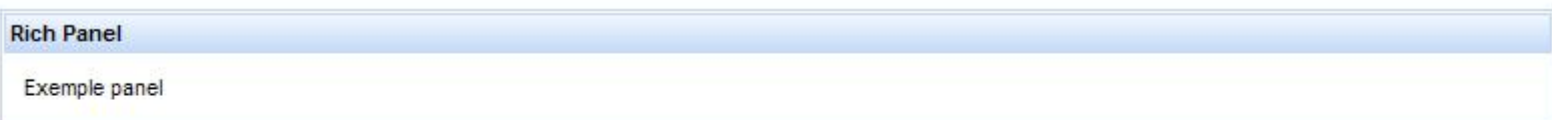
```
<rich:tabPanel>
  <rich:tab label="Client">
    composants pour le client
  </rich:tab>
  <rich:tab label="Commandes">
    composants pour commandes
  </rich:tab>
  <rich:tab label="Livraison">
    Composants pour livraison
  </rich:tab>
</rich:tabPanel>
```



VI-B - <rich:panel/>

Représente une zone rectangulaire qui peut inclure d'autres composants y compris d'autres panels. Il est possible de lui ajouter un en-tête avec un titre à l'aide d'un facet.

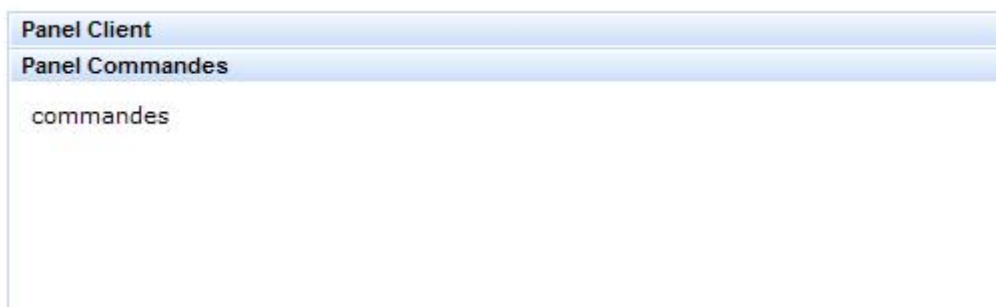
```
<rich:panel>
  <f:facet name="header">
    Rich Panel
  </f:facet>
  <h:outputText value="Exemple panel" />
</rich:panel>
```



VI-C - <rich:panelBar/>

C'est un ensemble de panels, lorsque l'un est ouvert, les autres restent fermés

```
<rich:panelBar height="150" width="500">
  <rich:panelBarItem label="Panel Client">
    Clients
  </rich:panelBarItem>
  <rich:panelBarItem label="Panel Commandes">
    commandes
  </rich:panelBarItem>
</rich:panelBar>
```



VI-D - <rich:simpleTogglePanel />

Représente un panel dont le contenu peut être affiché ou caché, grâce à un clic sur son titre.

Richfaces

Exemple de simpleTogglePanel, utiliser le lien en dans le coin haut/droit pour cacher ou afficher ce contenu.

VI-E - <rich:modalPanel/>

modalPanel est un composant en forme de popup modale, lorsque le modalPanel est affiché, la page parente devient inutilisable. Elle admet un facet header pour ajouter un en-tête ainsi qu'un facet controls pour ajouter un composant de contrôle dans l'en-tête.

modalPanel

```
<rich:modalPanel id="_panel_test" height="110" width="550">
  <f:facet name="header">Richfaces modalPanel</f:facet>
  <f:facet name="controls">
    <h:graphicImage value="close.png" style="cursor:pointer"
      onclick="Richfaces.hideModalPanel('_panel_test')" />
  </f:facet>
  <rich:panel style="border:0;height:100px">
    <h:outputText value="Bonjour!!" />
  </rich:panel>
</rich:modalPanel>
```

Exemple:

Exemple modalPanel

Richfaces modalPanel

Bonjour!!

Il existe trois manières de contrôler l'ouverture ou la fermeture du modalPanel:

Utiliser le code javascript:

ouvrir un modalPanel

```
<a4j:commandButton action="#{managedBean.action}"
value="Valider"
oncomplete="javascript:Richfaces.showModalPanel('_modalPanel_Id',{left:'auto',top:'auto'})" />
```

Fermer le modalPanel

```
<a href="javascript:Richfaces.hideModalPanel('_modalPanel_Id')">Close</a>
```

Utiliser le composant **rich:componentControl**

```

        <h:outputLink value="#" id="lien">
    <h:outputText value="Ouvrir modalPanel" />
    <rich:componentControl for="modalPanel_Id" attachTo="lien"
        operation="show" event="onclick" />
    </h:outputLink>


```

Utiliser les fonctions JS API show, hide:

```
<h:outputLink value="#" id="lien" onclick="#{rich:component('modalPanel_Id')}.hide()">Fermer</h:outputLink>
```

Il est également possible de passer des paramètres au modalPanel à partir de la page parente:

```
Richfaces.showModalPanel('panelId', {left: auto}, {param1: value1});
```

 **important:** Pour envoyer un formulaire à l'intérieur du modalPanel, il faut absolument respecter les règles suivantes:

- modalPanel doit avoir son propre form qui comporte ses éléments de type input
- modalPanel ne doit nullement se trouver à l'intérieur d'un form.

VI-F - <rich:dataTable/>

rich:dataTable apporte plusieurs fonctionnalités absentes dans le composant standard h:dataTable. En effet, rich:dataTable dispose des facets header et footer, il implémente les paramètres HTML rowspan et colspan. De plus, il peut contenir des sub-tables, et fournit des outils pour filtrer et ordonner les colonnes.

rich:dataTable

```

<rich:panel style="border:0;width:60%;text-align:center">
    <rich:panel>
        <f:facet name="header">
            Exemple dataTable
        </f:facet>
        <rich:dataTable cellpadding="0" cellspacing="0" border="0"
            var="list" value="#{testBean.persons}" id="table"
            style="text-align:center;" rows="5" width="100%">
            <f:facet name="header">
                <rich:columnGroup>
                    <rich:column colspan="3">
                        <h:outputText
                            value="Client(s) found(s): #{testBean.personsSize}" />

```


rich:dataTable

```
</rich:column>

<rich:column breakBefore="true">
  <h:outputText styleClass="headerText" value="Name" />
</rich:column>
<rich:column>
  <h:outputText styleClass="headerText" value="Company" />
</rich:column>
<rich:column>
  <h:outputText styleClass="headerText" value="Phone" />
</rich:column>
</rich:columnGroup>
</f:facet>
<rich:column sortBy="#{list.name}">
  <h:outputText value="#{list.name}" />
</rich:column>
<rich:column filterEvent="onkeyup" filterBy="#{list.company}">
  <h:outputText value="#{list.company}" />
</rich:column>
<rich:column>
  <h:outputText value="#{list.phone}" />
</rich:column>
</rich:dataTable>
<rich:datascroller for="table"></rich:datascroller>
</rich:panel>
</rich:panel>
```

Le résultat de l'exemple:

Exemple dataTable	
Client(s) found(s): 10	
Name	Company
⬆	<input type="text"/>
Client 0Name	company for Client 0
Client 1Name	company for Client 1
Client 2Name	company for Client 2
Client 3Name	company for Client 3
Client 4Name	company for Client 4

<< < 1 2 > >>

rich:dataTable dispose à l'instar d'autres composants d'itération de données, de nombreux événements javascript pouvant lancer une requête Ajax avec l'aide du composant a4j:support. Les événements disponibles pour rich:dataTable sont :

- onclick
- ondblclick
- onkeydown

- onkeypress
- onkeyup
- onmousedown
- onmousemove
- onmouseout
- onmouseover
- onmouseup
- onRowClick
- onRowDbClick
- onRowMouseDown
- onRowMouseMove
- onRowMouseOut
- onRowMouseOver
- onRowMouseUp

L'exemple suivant présente l'utilisation de l'événement **onRowClick** pour afficher le contenu de la ligne sélectionnée dans une pop-up, ou récupérer l'objet correspondant dans le but de réaliser un traitement coté serveur.

```
<rich:dataTable border="0" var="list"
value="#{managedBean.dataTableRows}" id="table">
<a4j:support event="onRowClick"
action="#{managedBean.processRowUpdate}"
oncomplete="javascript:Richfaces.showModalPanel('_panel_Row_Details',{left:'auto', top:'auto'})">
<f:setPropertyActionListener value="#{list}"
target="#{managedBean.selectedRow}" />
</a4j:support>

<!-- déclaration des colonnes -->
</rich:dataTable>
```



*Notez l'utilisation de **f:setPropertyActionListener** pour envoyer l'objet correspondant à la ligne en cours au serveur lors de la requête Ajax et dont la valeur est la propriété **selectedRow** du bean. D'autres méthodes pour retrouver la ligne courante coté serveur sont expliquées dans la **FAQ JSF**.*

VI-G - <rich:DataTableScroller/>

Ce composant est associé à rich:dataTable pour faire la pagination de la table à l'aide des requêtes Ajax

```
<rich:datascroller id="scroller1" for="table" reRender="scroller2" align="center" renderIfSinglePage="false" />
```

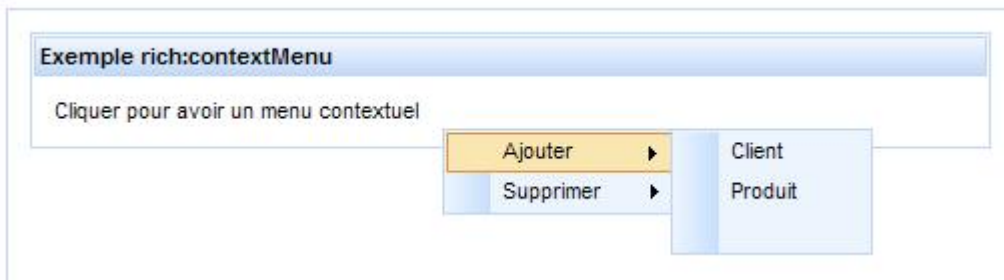
VI-H - <rich:contextMenu />

Le contextMenu est un menu que l'on peut attacher à un autre composant JSF, il s'active par un événement JavaScript. L'exemple suivant présente un contextMenu attaché au composant rich:panel.

```
<rich:panel id="panel">
<f:facet name="header">
<h:outputText value="Exemple rich:contextMenu" />
</f:facet>
<h:outputText value="Cliquer pour avoir un menu contextuel" />
</rich:panel>
<rich:contextMenu event="onclick" attachTo="panel" submitMode="ajax">
<rich:menuGroup value="Ajouter">
```

```
<rich:menuItem value="Client"></rich:menuItem>
<rich:menuItem value="Produit"></rich:menuItem>
<rich:menuItem></rich:menuItem>
</rich:menuGroup>
<rich:menuGroup value="Supprimer">
  <rich:menuItem value="Client"></rich:menuItem>
  <rich:menuItem value="Produit"></rich:menuItem>
</rich:menuGroup>
</rich:contextMenu>
</rich:panel>
```

Ce qui donne :



Le contextMenu contient un ou plusieurs composants **menuItem** groupés à l'aide de **menuGroup**, les actions sur les menus peuvent se faire à l'aide des attributs **action** et **actionListener**. Le paramètre **submitMode** de rich:contextMenu décrit la méthode de l'envoi de l'action au serveur : **ajax**, **server**, **none**.

Il est très courant d'utiliser rich:contextMenu avec dataTable pour réaliser des traitements sur une ligne particulière. Pour éviter d'avoir un nombre important de balises rendus dans la dataTable, il est intéressant d'utiliser contextMenu avec **rich:componentControl**. L'exemple suivant présente l'utilisation du contextMenu avec dataTable et componentcontrol.

La déclaration de dataTable avec componentControl :

```
<rich:dataTable var="list" value="#{testBean.persons}" id="table"
  rows="15">
  <rich:column>
    <f:facet name="header">
      <h:outputText value="Nom" />
    </f:facet>
    <h:outputText value="#{list.name}" />
  </rich:column>
  <rich:column>
    <f:facet name="header">
      <h:outputText value="Entreprise" />
    </f:facet>
    <h:outputText value="#{list.company}" />
  </rich:column>
  <rich:componentControl event="onRowClick" for="menu" operation="show">
    <f:param value="#{list.name}" name="userName" />
    <f:param value="#{list.company}" name="company" />
  </rich:componentControl>
</rich:dataTable>
```

Ensuite le contextMenu :

```
<rich:contextMenu attached="false" id="menu" submitMode="ajax">
  <rich:menuItem ajaxSingle="true">
    <b>{userName}</b>
  </rich:menuItem>
  <rich:menuGroup value="Operations">
    <rich:menuItem ajaxSingle="true" reRender="table">
```

```

    Modifier <b>{userName}</b>
  </rich:menuItem>
  <rich:menuItem ajaxSingle="true" reRender="table">
    Supprimer <b>{userName}</b>
  </rich:menuItem>
</rich:menuGroup>
</rich:contextMenu>

```

Ce qui donne :

Exemple rich:contextMenu

Nom	Entreprise
Paul0Name	company for Client 0
Paul1Name	company for Client 1
Paul2Name	company for Client 2
Paul3Name	company for Client 3
Paul4Name	company for Client 4
Paul5Name	company for Client 5
Paul6Name	company for Client 6
Paul7Name	company for Client 7
Paul8Name	company for Client 8
Paul9Name	company for Client 9



Notez l'utilisation de la syntaxe **JSON** pour passer la variable de la ligne en cours au `contextMenu` :

```
<f:param value="#{list.name}" name="userName" />
```

```
Modifier <b>{userName}</b>
```

Pour transmettre la variable du menu au managedBean, on peut utiliser le composant **actionparam**

```

<rich:menuItem ajaxSingle="true" reRender="table"
  actionListener="#{bean.modifierUser}">
  Modifier {userName}
<a4j:actionparam name="user_name" assignTo="#{bean.selectedUserName}"
  value="{userName}" />

```

VI-I - <rich:jQuery />

Richfaces prend en charge le framework javascript **jQuery**, le composant rich:jQuery permet d'utiliser la puissance du framework jQuery pour des composants JSF/Richfaces. Pour plus d'informations sur l'utilisation de jQuery avec d'autres libraires, consultez les **exemples suivants**.

L'exemple ci-dessous décrit l'utilisation de jQuery dans une dataTable, les lignes successives de la dataTable n'ont pas le même style, nous souhaitons, lors du passage de la souris sur une ligne changer le style de cette ligne, mais, retrouver l'ancien style lorsque la souris est en dehors de la ligne.

```
<rich:dataTable rowClasses="even-row ,odd-row">
<!-- dataTable avec lignes paires et impaires de couleurs différentes-->
</rich:dataTable>
```

jQuery utilise ce qu'on appelle un selector pour désigner le composant sélectionné pour le traitement javascript par une fonction jQuery. Ici le selector est la ligne de la table: *table tr*.

```
<rich:jQuery selector="#table tr"
query="mouseover(function(){jQuery(this).addClass('active-row')}} " />
<rich:jQuery selector="#table tr"
query="mouseout(function(){jQuery(this).removeClass('active-row')}} " />
```

VI-J - <rich:ajaxValidator/>

Ce composant existe depuis la 3.2.2.GA, il est destiné à valider un champ input avec une requête Ajax qui saute toutes les phases du cycle JSF sauf la phase de validation et la phase render response. Il dispose des mêmes fonctionnalités que **aj:support**. Par défaut, **ajaxSingle** est égal à true ce qui veut dire que seul le composant en question est envoyé dans la requête.

```
<h:panelGrid columns="3">
<h:outputText value="Login:" />
<h:inputText value="#{managedBean.login}" id="login" required="true">
<f:validateLength minimum="2" maximum="8" />
<rich:ajaxValidator event="onblur" />
</h:inputText>
<rich:message for="login" />
<h:outputText value="Mot de passe:" />
<h:inputText value="#{managedBean.mdp}" id="mdp" required="true">
<f:validateLength minimum="6" maximum="12" />
<rich:ajaxValidator event="onblur" />
</h:inputText>
<rich:message for="mdp" />
</h:panelGrid>
```

Ce composant permet aussi d'utiliser la validation hibernate.

```
@NotEmpty
@Length(min=2,max=8)
private String login;
```

```
<h:inputText value="#{managedBean.login}" id="login">
<rich:ajaxValidator event="onblur" />
</h:inputText>
```

VII - Améliorer la performance des applications Richfaces

Nous avons passé en revue les principaux composants de Richfaces, certains d'entre eux laissent au développeur le soin de spécifier les éléments à envoyer au serveur pour le traitement dans les différentes phases de JSF. Lorsque l'interface utilisateur contient un nombre important de données, et, lorsqu'il existe beaucoup d'interactions entre le client et le serveur, il devient important de n'envoyer au serveur que les composants utiles pour la requête. De plus, dans certains cas, seulement quelques phases sont nécessaires à l'exécution de la requête. Pour cela, l'utilisation à bon escient de quelques attributs de composants Richfaces permet un gain notable de temps de réponse.

VII-A - Contrôler la file d'attente des requêtes Ajax

Si on utilise l'envoi de requêtes Ajax avec l'événement onkeyup, par exemple, il se peut que l'utilisateur écrit tellement vite que le serveur s'inonde de requêtes à traiter. Pour éviter ce genre de désagréments, les composants Richfaces disposent d'attributs importants :

- 1 **eventsQueue** : permet d'envoyer chaque requête dans une liste d'attente dans le cas où la requête antérieure ne s'est pas complètement terminée.
Pour le faire, il suffit de donner le nom de la file d'attente à l'attribut eventsQueue au composant Richfaces envoyant la requête. L'exemple suivant illustre l'utilisation de eventsQueue avec a4j:support :

```
<h:inputText value="#{managedBean.nom}">
  <a4j:support event="onkeyup" eventsQueue="maFile"
    reRender="composant_id" />
</h:inputText>
```

Pour mieux comprendre le fonctionnement de cet attribut, il est utile de noter que la file d'attente ne peut contenir qu'un seul élément, donc, si l'utilisateur saisit un caractère, puis, saisit rapidement 4 autres caractères avant que la première requête ne soit terminée, nous ne pouvons pas avoir une liste de cinq requêtes, mais, les quatre derniers caractères sont groupés dans une requête et mis dans une file d'attente. Donc, toute requête intermédiaire mise dans une file d'attente annulera la précédente qui n'a pas été encore envoyée au serveur ou n'a pas fini sa réponse au client.

- 2 **requestDelay** : permet de définir le temps en millisecondes de mise en attente de la requête, cette dernière ne sera prête à l'envoi qu'à la fin de ce laps de temps. L'exemple suivant bloque l'envoi de la requête pendant trois secondes :

```
<a4j:commandButton action="#{managedBean.action}" value="Valider"
  requestDelay="3000" />
```

- 3 **ignoreDupResponses** : lorsque sa valeur est à true, il permet d'ignorer la requête courante, si une requête du même composant est dans file d'attente. On peut noter que cela suppose que la requête originale a déjà été traitée et que la réponse ne s'est pas entièrement réalisée.

```
<a4j:commandButton value="Valider" ignoreDupResponses="true"
  reRender="zone" />
```

VII-B - Contrôler l'exécution des phases JSF

Chaque requête Ajax, comme toute requête JSF suit le cycle de vie JSF et passe par les six phases de ce cycle. Toutefois, l'exécution des six phases n'est pas toujours nécessaire au bon fonctionnement de la requête. Par conséquent, l'utilisation des composants et attributs suivants peut se révéler très utile.

- 1 **bypassUpdates** : à utiliser pour la validation d'un composant, il permet, en effet, de ne pas invoquer les phases Update Model et Invoke Application, ce qui accroît le temps de réponse.
- 2 **ajaxSingle** : permet de limiter l'exécution des phases JSF seulement pour le composant en cours, si sa valeur est égale à false tous les composants présents dans la requête passent par les six phases JSF.

- 3 **a4j:region** : permet de réduire l'ensemble de composants traités coté serveur pour améliorer la performance, seulement les composants présents à l'intérieur de a4j:region seront traités.
- 4 **renderRegionOnly** : cet attribut de a4j:region permet de limiter la mise à jour du client seulement à la région courante à partir de laquelle la requête a été envoyée, lorsque des composants externes à la région sont définis dans **reRender** ils ne seront pas mis à jours.
- 5 **immediate** : cet attribut disponible également dans les composants de bases JSF permet d'éviter les phases conversions/validations.



Pour suivre les phases d'une requête Ajax/JSF, il est possible de journaliser les phases JSF de l'application. Veuillez trouver l'explication dans la [FAQ JSF](#)

VIII - Richfaces skins

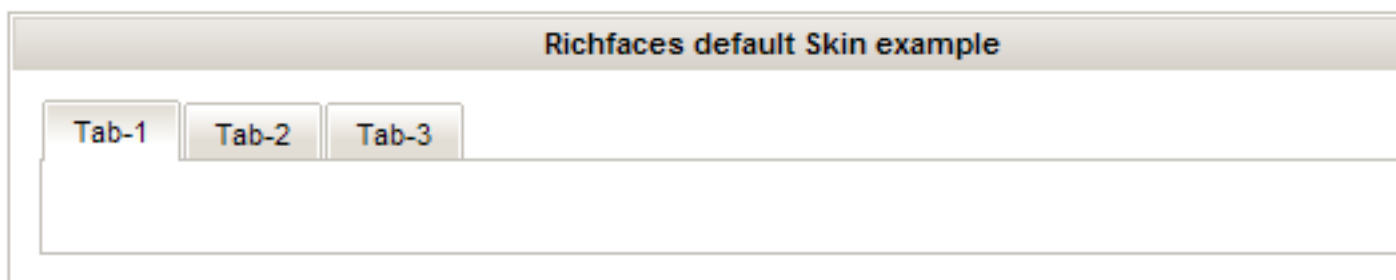
VIII-A - Présentation

Richfaces propose un système de plug-in de styles appelé **skin** pour habiller les composants Richfaces et JSF, l'ensemble de ses plug-ins s'enrichie à chaque nouvelle version de Richfaces.

Les skins disponibles par défaut sont :

- DEFAULT
- plain
- emeraldTown
- blueSky
- wine
- japanCherry
- ruby
- classic
- deepMarine
-

Le style DEFAULT est utilisé par défaut, l'exemple suivant présente les composants rich:panel et rich:tabPanel :

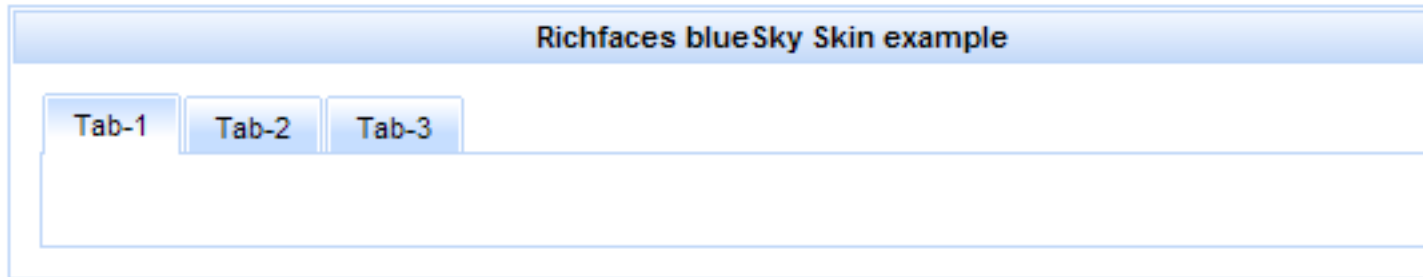


VIII-B - Utiliser un skin

Pour utiliser un style il suffit de donner son nom au paramètre du context **org.richfaces.SKIN**

```
<context-param>
  <param-name>org.richfaces.SKIN</param-name>
  <param-value>blueSky</param-value>
</context-param>
```

Ce qui donne:



VIII-C - Personnaliser un skin

Les styles fournis par défaut sont définis dans un fichier properties placé dans le dossier /META-INF/skins de l'archive Richfaces-impl. Pour modifier un style :

- Désarchiver Richfaces-impl-3.2.x.jar et prendre le fichier de properties dans le dossier /META-INF/skins
- Modifier le fichier properties et le sauvegarder dans le classpath de l'application sous un autre nom

A titre d'exemple, nous allons modifier la couleur du border de rich:panel dans le skin blueSky. Pour cela, on ouvre le fichier **blueSky.skin.properties**, et on modifie la clé **panelBorderColor**.

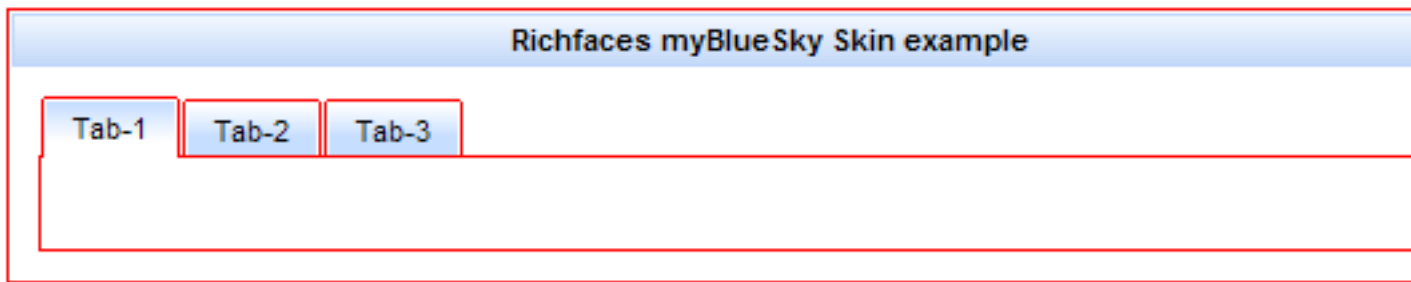
```
#panelBorderColor=#BED6F8
panelBorderColor= #FF0000
```

On renomme le fichier en **myBlueSky.skin.properties** et on le sauvegarde dans un package de l'application.

- Donner le nom du nouveau skin au paramètre **org.richfaces.SKIN**

```
<context-param>
  <param-name>org.richfaces.SKIN</param-name>
  <param-value>myBlueSky</param-value>
</context-param>
```

L'exemple précédent avec myBlueSky skin:



Depuis la version 3.2.1 le processus est devenu plus simple, il suffit de créer un nouveau fichier de skins et déclarer le skin à modifier comme ceci:

```
baseSkin=blueSky
panelBorderColor= #FF0000
```

VIII-D - Format XCSS

Richfaces skinning se base sur le format XCSS pour appliquer le style aux composants, XCSS est un format XML de CSS qui ajoute des fonctionnalités supplémentaires processus d'application de classes de style. Le fichier XCSS ajoute des propriétés CSS aux classes à l'aide des tags **u:selector** et **u:style**.

Exemple de fichier XCSS:

```
<u:selector name=".rich-component-name">
    <u:style name="background-color" skin="additionalBackgroundColor" />
    <u:style name="border-color" skin="tableBorderColor" />
    <u:style name="border-width" skin="tableBorderWidth" />
    <u:style name="border-style" value="solid" />
</u:selector>
```

Le code ci-dessus sera parsé en:

```
.rich-component-name {
background-color: additionalBackgroundColor; /*the value of the constant defined by your skin*/
border-color: tableBorderColor; /*the value of the constant defined by your skin*/
border-width: tableBorderWidth /*the value of the constant defined by your skin*/
border-style: solid;
}
```

additionalBackgroundColor correspond à une clé dans le fichier properties du skin.

VIII-E - Utiliser les skins Richfaces pour les composants HTML

Richfaces propose également des styles évolués pour les composants HTML standards pour améliorer le rendu de ces composants et rendre ces derniers cohérents avec l'ensemble des composants Richfaces. Il existe deux niveaux de skinning avec Richfaces:

- **Standard** : propose la personnalisation de propriétés de style basique, il est appliqué pour les navigateurs : Internet Explorer 6, Opera, Safari.
- **Étendu** : étend le niveau standard et propose davantage de propriétés de style qui sont appliqués pour les navigateurs avec des capacités visuelles riches. Les navigateurs supportés par le niveau étendu sont : Fire Fox Mozilla et Internet Explorer 7.

Les éléments HTML affectés par Richfaces skinning sont :

- input
- select
- textarea
- keygen
- isindex
- legend
- fieldset
- hr
- a

Pour activer le skinning Richfaces, il est nécessaire de définir deux paramètres du contexte de l'application web :

- **org.richfaces.CONTROL_SKINNING** : prend les valeurs **enable/desable**, il implique l'utilisation des classes de styles aux éléments par leurs noms et leurs types. De ce fait, le développeur n'intervient pas dans l'application du style, les éléments input par exemple auront automatiquement le style du skin défini.
- **org.richfaces.CONTROL_SKINNING_CLASSES** : prend les valeurs **enable/desable**, permet lorsque la valeur est égale à enable d'appliquer à un type de composants une classe de style particulière. Richfaces applique par ce moyen les classes aux composants de bases ayant la classe de style **rich-container** comme :

```
.rich-container select {  
}  
  
.rich-container hr {  
}
```

Les skins s'appliquent également à d'autres composants dont le nom de la classe CSS correspond à la règle: **rich-nom-type** comme:

```
.rich-select {  
}  
.rich-input-text {  
}
```

L'exemple suivant présente les éléments inputText inputTextarea et commandButton dans le skin blueSky

Richfaces Skinning example

Login

Nom

Commentaire

Click

IX - Astuces et questions fréquentes

IX-A - [rich:TabPanel] comment définir l'onglet courant :

On utilise le paramètre **selectedTab** de rich:tabPanel

```

<rich:tabPanel selectedTab="#{managedBean.selectedTab}">
  <rich:tab label="Tab 1" name="t1">
    <h:outputText value="tab 1" />
  </rich:tab>
  <rich:tab label="Tab 2" name="t2">
    <h:outputText value="tab 2" />
  </rich:tab>
</rich:tabPanel>

```

Dans le managed bean, on définit dynamiquement l'onglet courant :

```
setSelectedTab("t2") ;
```

IX-B - [rich:TabPanel] Comment réaliser une navigation en cliquant sur un onglet

En utilisant le paramètre **action**

```

<rich:tabPanel switchType="server">
  <rich:tab label="Tab 1" name="t1" action="index">
  </rich:tab>
</rich:tabPanel>

```

Et pour la navigation

```

<navigation-rule>
  <from-view-id>/source.jsp</from-view-id>
  <navigation-case>
    <from-outcome>index</from-outcome>
    <to-view-id>/index.jsp</to-view-id>
  </navigation-case>

```

</navigation-rule>

IX-C - [rich:TabPanel] Comment définir son propre style des onglets

En redéfinissant les classes de style suivantes :

- contentClass,
- activeTabClass,
- inactiveTabClass,
- tabClass.

```
<rich:tabPanel contentClass="myrich-tabpanel-content"
  activeTabClass="myrich-tab-active"
  inactiveTabClass="myrich-tab-inactive" tabClass="myrich-tabpanel">
</rich:tabPanel>
```

IX-D - [rich :TabPanel] Comment créer dynamiquement TabPanel

En utilisant JSTL: c:forEach,

En java:

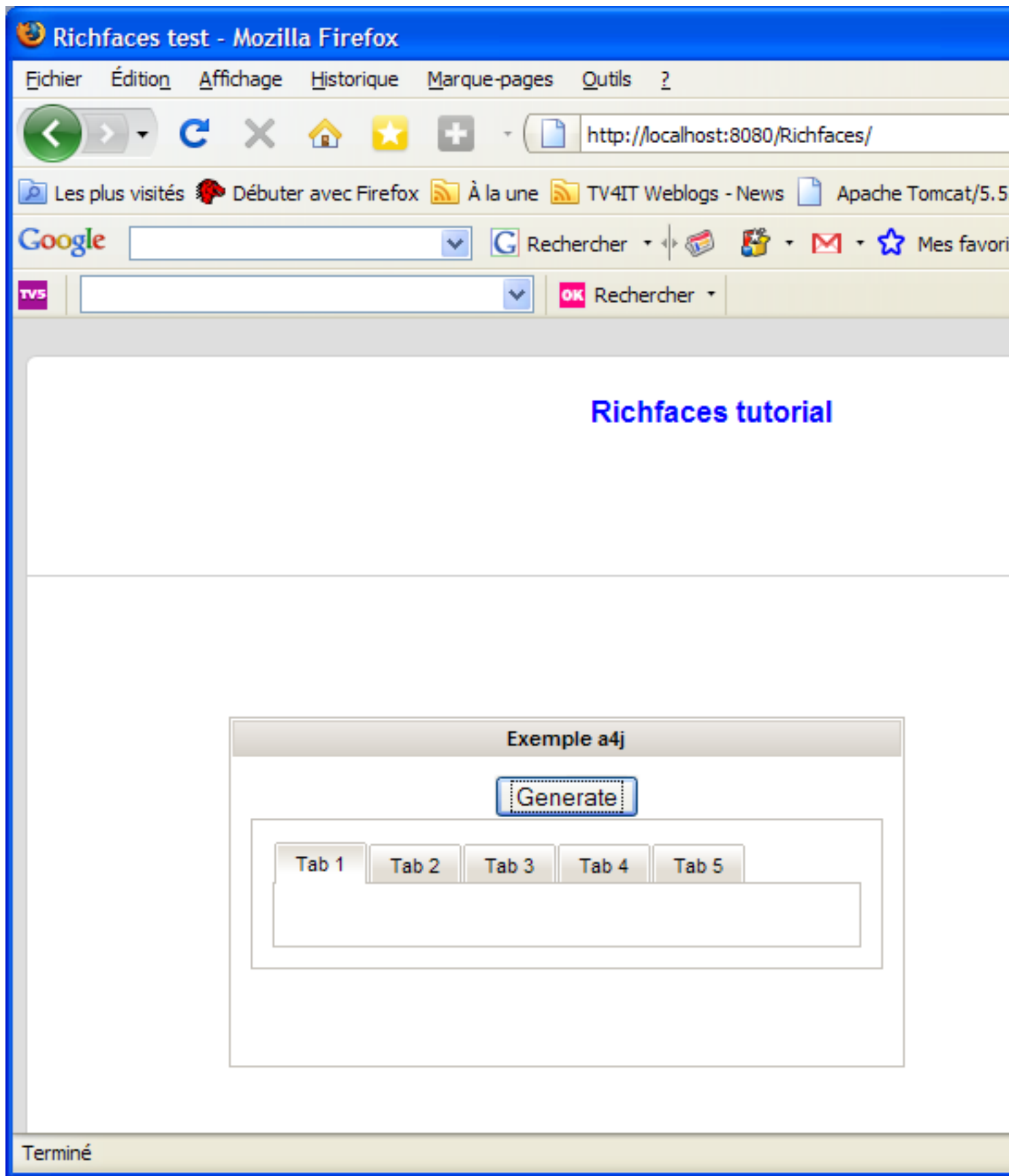
```
<a4j:commandButton value="Generate"
  actionListener="#{managedBean.generateTabs}" reRender="tabs" />
<rich:panel binding="#{managedBean.panel}" id="tabs">
</rich:panel>
```

Dans ManagedBean.java

```
public void generateTabs(ActionEvent e) {
    Application application = FacesContext.getCurrentInstance().getApplication();
    HtmlTabPanel tabPanel = (HtmlTabPanel) application.createComponent(HtmlTabPanel.COMPONENT_TYPE);
    tabPanel.setSwitchType("ajax");
    for (int i = 0; i < 5; i++) {
        HtmlTab tab = (HtmlTab) application.createComponent(HtmlTab.COMPONENT_TYPE);
        tab.setLabel("Tab " + (i + 1));
        tabPanel.getChildren().add(tab);
    }

    panel.getChildren().clear();
    panel.getChildren().add(tabPanel);
}
```

Ce qui donne :



IX-E - [rich:dataTable] : comment afficher un modalPanel à la fin de l'exécution d'une action sur une ligne

En utilisant `a4j:support` à l'intérieur de la `dataTable` avec l'événement `onRowClick` ou `onRowDbClick`

```
<a4j:support event="onRowDbClick" reRender="panel_details " action="#{managedBean.doAction}"
oncomplete="javascript:Richfaces.showModalPanel('_panel',{left:'auto', top:'auto'})" />
```

IX-F - [rich :dataTable] : comment définir un style différent pour deux lignes successives

Il suffit de définir deux classes de style dans le paramètre `rowClasses`

```
<rich:dataTable rowClasses="even-row ,odd-row" />
```

IX-G - [rich :dataTable] : comment filtrer les données dans le managed Bean

Utiliser le paramètre **filterMethod** :

```
<rich:column id="status" filterMethod="#{managedBean.filterStatus}"
filterValue="#{managedBean.filteredStatus}">
<f:facet name="header">
<h:selectOneMenu value="#{managedBean.filteredStatus}">
<f:selectItems value="#{managedBean.status}" />
<a4j:support event="onchange" reRender="table_id" />
</h:selectOneMenu>
</f:facet>
<h:outputText value="#{list.status}" />
</rich:column>
```

Dans le managed Bean:

```
private String filteredStatus = "";

public boolean filterStatus(Object current) {
    DataTableBean bean = (DataTableBean) current;
    if (filteredStatus.length() == 0) {
        return true;
    }
    if (bean.getStatus() != null &&
        bean.getStatus().toLowerCase().startsWith(filteredStatus.toLowerCase())) {
        return true;
    } else {
        return false;
    }
}
```

IX-H - [rich:dataTable] : comment réduire la taille du champ texte généré par filterBy

Utiliser la classe **rich-filter-input**

```
<rich:column headerClass="userColumn" filterEvent="onkeyup"
filterBy="#{list.nom}">
<h:outputText value="#{list.nom}" />
</rich:column>
```

dans le CSS:

```
.userColumn .rich-filter-input {
    width: 50px;
}
```

IX-I - [rich :dataTable] : comment cacher/afficher dynamiquement une colonne

Utilisation de l'attribut **rendered**

```
<rich:column id="adress_id" sortBy="#{list.adresse}"
    rendered="#{managedBean.cellRendered['adress_id']}">
    <f:facet name="header">
        <h:panelGroup layout="block">
            <h:outputText value="Adresse" />
            <h:commandLink value="Cacher">
                <a4j:actionparam name="cacher" value="false"
                    assignTo="#{managedBean.cellRendered['adress_id']}" />
            </h:commandLink>
        </h:panelGroup>
    </f:facet>
    <h:outputText value="#{list.adresse}" />
</rich:column>
```

Dans le managed bean

```
public ManagedBean() {

    cellRendered.put("adress_id", Boolean.TRUE);

}
```

Pour réafficher la colonne:

```
<h:commandLink value="Afficher">
    <a4j:actionparam name="afficher" value="true"
        assignTo="#{managedBean.cellRendered['adress_id']}" />
</h:commandLink>
```

IX-J - [rich :dataTable] [rich :dataScroller] : comment sélectionner la page courante avec un selectOneMenu :

Utiliser le paramètre **page** du dataScroller, exemple dataScroller à l'intérieur de la table :

```
<f:facet name="footer">
    <h:panelGroup>
        <h:selectOneMenu value="#{managedBean.page}">
            <f:selectItems value="#{managedBean.pages}" />
            <a4j:support reRender="autre_scroller" event="onchange" />
        </h:selectOneMenu>
        <rich:datascroller pageIndexVar="pageIndex" pagesVar="pages"
            for="table" id="footer_scroller" reRender="table"
            page="#{managedBean.page}">
            <f:facet name="pages">
                <h:outputText value="#{pageIndex} / #{pages}" />
            </f:facet>
        </rich:datascroller>
```

```
</h:panelGroup>
</f:facet>
```

IX-K - [rich:dataTable] [rich:dataScroller] : comment mettre à jour le dataScroller après l'utilisation d'un filtre:

A partir de la 3.2.2.GA rich:dataTable dispose du paramètre reRender qui permet de rafraîchir un composant suite à une requête Ajax à partir de la dataTable

```
<a4j:outputPanel id="scroller">
  <rich:datascroller for="table" reRender="table" align="center" />
</a4j:outputPanel>
<rich:dataTable reRender="scroller" />
```

IX-L - [rich:modalPanel] comment ajouter un scroller à la modalPanel.

Redéfinir les classes comme décrit ci-dessous :

```
<f:subview xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:a4j="http://richfaces.org/a4j"
  xmlns:rich="http://richfaces.org/rich">

  <style type="text/css">
    .dr-mpnl-pnl {
      overflow: auto !important
    }

    .rich-mpnl-body {
      height: 92%;
      width: 98%;
    }

    .rich-mpnl-body .content {
      padding: 0px;
    }
  </style>
  <rich:modalPanel id="_panel">
  </rich:modalPanel>
</f:subview>
```

IX-M - [rich:modalPanel] Comment afficher un modalPanel au chargement d'une page

Utiliser le paramètre **showWhenRendered**, il permet au modalPanel de s'afficher seulement si il est à true.

```
<rich:modalPanel id="_panel" showWhenRendered="true" />
```

IX-N - [rich:modalPanel] comment afficher les messages d'erreurs de validation dans un modalPanel

Voici l'exemple d'une page de login avec deux champs obligatoires, l'attribut showWhenRendered permet au modalPanel de s'afficher seulement si il y'a des messages d'erreurs dans le context.

```
<h:panelGrid columns="2">
  <h:outputText value="Login" />
```



```
<h:inputText value="#{testBean.login}" required="true" />
<h:outputText value="Mot de passe" />
<h:inputText value="#{testBean.passwd}" required="true" />
</h:panelGrid>
<a4j:commandButton action="#{testBean.connect}" value="Entrer"
type="submit"
oncomplete="javascript:Richfaces.showModalPanel('_panel_error',{left:'auto',top:'auto'})" />

<rich:modalPanel id="_panel_error"
showWhenRendered="#{facesContext.maximumSeverity !=null}">
<f:facet name="header">Login errors</f:facet>
<rich:panel style="border:0;height:100px">
<rich:messages layout="list">
<f:facet name="errorMarker">
<h:graphicImage value="/images/error.gif" />
</f:facet>
</rich:messages>
</rich:panel>
</rich:modalPanel>
```

IX-O - Comment gérer l'expiration de session avec Richfaces


Richfaces dispose d'une fonction JavaScript lancée lors de l'expiration de la session, nous allons l'utiliser pour afficher un modalPanel après l'expiration de session.

Définition du temps d'expiration de session dans le web.xml

```
<session-config>
<session-timeout>20</session-timeout>
</session-config>
```

On définit le paramètre du context suivant

```
<context-param>
<param-name>org.ajax4jsf.handleViewExpiredOnClient</param-name>
<param-value>true</param-value>
</context-param>
```

 **A4J.AJAX.onExpired** fonctionnait bien avant le passage à la version 3.2.0 de Richfaces sans ajouter ce paramètre, depuis la 3.2.0 et le passage à JSF 1.2 il y'a eu une incompatibilité avec la gestion de l'expiration de session de JSF, le problème a été corrigé dans la version 3.2.2.GA en ajoutant ce paramètre du context.

Le modalPanel suivant doit être inclus dans toutes les pages de l'application :

```
<rich:modalPanel id="sessionExpiredPanel">
<f:facet name="header">Session expired</f:facet>
<rich:panel style="border:0;text-align:center;">
<h:outputText value="Votre session a expiré!" />
</rich:panel>
</rich:modalPanel>
```

On définit un appel Ajax à une fréquence supérieure au temps du timeout de quelques millisecondes, et on affiche le modalPanel sur un événement d'expiration de session.

```
<a4j:region>
<a4j:form>
<a4j:poll id="sessioncheck" interval="1250000"
reRender="sessioncheck" />
```

```
</a4j:form>
<script type="text/javascript">
    A4J.AJAX.onExpired = function(loc,expiredMsg){
        Richfaces.showModalPanel('sessionExpiredPanel',{left:'auto',top:'auto'});
    }
</script>
</a4j:region>
```

IX-P - [rich:comboBox] comment sélectionner un élément par son id ?

Il n'est pas possible de récupérer l'id d'un selectItem dans une comboBox, car, pour permettre l'auto-completion coté client, le composant est de type Input et pas select. On peut le considérer comme un inputText évolué. Toutefois, la dernière version de Richfaces permet de donner à ce composant la valeur d'un objet et pas seulement un type String.

IX-Q - [rich:suggestionBox] comment récupérer l'id d'un élément sélectionné?

Utiliser **a4j:support** et **setPropertyActionListener**

```
<rich:suggestionbox id="suggestionCompanyId" for="companySuggest"
tokens="," rules="none"
suggestionAction="#{managedBean.autocompleteCompany}" var="result"
fetchValue="#{result.companyName}" rows="0" first="0">
<a4j:support event="onselect" action="#{managedBean.someAction}">
    <f:setPropertyActionListener value="#{result.companyId}"
        target="#{managedBean.selectedCompanyId}" />
</a4j:support>
</rich:suggestionbox>
```

X - Remerciements

Je tiens à remercier :

- **djo.mos** pour son encouragement.
- **romaintaz** pour sa relecture.
- **OButterlin** pour sa relecture.
- **ridekick** pour sa relecture.