

Contrôle XSL d'autocomplétion dans les applications XMLRAD

par Sylvain James ([Site Developpez de Sylvain James](#)) ([Blog](#))

Date de publication : 17/11/2007

L'objectif de cet article est de montrer combien un composant XSL peut faciliter l'implémentation de fonctionnalités complexes, telles que l'autocomplete au sens Web 2.0.

L'autocomplete est une fonctionnalité Web 2.0 souvent plebiscitée par les utilisateurs. Vous renseignez une valeur à rechercher dans une boîte de saisie, puis cette boîte affiche une liste de réponses respectives.

Sur le plan technique, il sera fait appel en arrière plan à des requêtes XHR ainsi qu'à du javascript avancé.

Mais toute complexité est masquée par l'implémentation du composant XSL présenté ci-dessous.

I - Présentation

I-A - Un composant autocomplete ?

I-B - La Yahoo Lib

I-C - Pré-requis

II - Architecture

II-A - Diagramme de séquence

III - Mise en oeuvre du composant XSL Autocomplete

III-A - Déclaration du namespace yui

III-B - Ajout des headers javascript

III-C - Insertion du composant XSL

III-D - Ajout du XMLService fournisseur de données

IV - Paramètres du composant

IV-A - Présentation

IV-B - Référence paramètres

IV-C - Exemple : Mise à jour de champs lors de la sélection

IV-D - Exemple : Saisie multiple, choix de destinataires email

IV-E - Exemple : Recherche et vérification d'adresse via la géolocalisation de GoogleMaps

V - Conclusion

V-A - Générales


V-B - Améliorations

V-C - Et les sources... ?

I - Présentation

I-A - Un composant autocomplete ?

Intégrer un composant XSL d'autocomplétion dans vos applications XMLRAD, tel est l'objectif de ce tutorial.

Si vous voulez rapidement vous faire une idée concrète du résultat, visitez les démos suivante :  **Démos Yahoo Library**, ou les exemples ici dans la section IV (-C, -D et -E).

Mais si vous vous attardez sur le code source javascript que vous devez manipuler, vous risquez d'être effrayé... Non pas que les killers de yahoo aient mal codé, mais justement, le niveau en javascript est tel qu'un débutant aura du mal à être à l'aise.

J'ai **encapsulé cette complexité dans un composant XSL** que je vous propose de découvrir. Il n'implémente pas encore la totalité des propriétés et méthodes du composant original, mais évolue régulièrement. A l'inverse certaines fonctionnalités jugées utiles ont été ajoutées. Si vous êtes pressés, allez directement à la section III qui explique pas à pas l'insertion du composant autocomplete.

Nous mettrons en oeuvre le composant à travers 3 exemples sympathiques :

- Le plus simple qui interroge une liste de pays avec la base Training.
- Un peu plus compliqué, on renseigne un champ mail de destinataires.
- Enfin très sympa, une boîte de vérification d'adresse s'appuyant sur les services google maps.

I-B - La Yahoo Lib

Le framework javascript de Yahoo (<http://developer.yahoo.com/yui/>) a mes faveurs depuis un bout de temps déjà, pour les raisons suivantes :

- C'est un des seuls à être bien documenté et fourni avec des exemples (même si on aimerait toujours plus d'exemples !).
- Il est relativement facile à intégrer dans une application.
- Il est OpenSource.
- Yahoo étant utilisateur de son propre framework, c'est plutôt rassurant pour les corrections et évolutions à venir.
- La YUILib a donné lieu à une déclinaison sous la forme d'un autre projet particulièrement bluffant : **Ext.js** anciennement nommé la YUI-Ext.

I-C - Pré-requis

Si vous suivez pas à pas ce tutoriel, il n'y a aucune raison pour que vous ne puissiez obtenir les mêmes résultats !

Maintenant si vous souhaitez comprendre les fondations, vous devrez avoir un minimum de connaissances en XSL et une pratique assez régulière de l'environnement XMLRAD.

Pour ce qui concerne javascript, aucune inquiétude sauf si vous êtes un technicien curieux qui cherche à comprendre les sources de la Yahoo Lib... Là vous aurez besoin d'un level JEDI :-)

II - Architecture

La lecture de cette section n'est pas obligatoire pour la mise en oeuvre du composant, mais il est intéressant de comprendre comment le composant Autocomplete de Yahoo fonctionne.

On peut découper le processus en six étapes :

- 1 La zone d'édition intégrant une détection à la saisie
- 2 Une fois un délai imparti après le dernier caractère saisi, déclenchement de la recherche
- 3 L'invocation d'une action http afin de récupérer un résultat
- 4 La production d'un résultat (XML) par un processus externe
- 5 La mise en forme de ce résultat sous forme de liste
- 6 L'action à entreprendre lorsqu'un des résultats est sélectionné

II-A - Diagramme de séquence

La mise en oeuvre de ces étapes impliquent une connaissance avancée du javascript, heureusement implémentée par la librairie YUI.

Le diagramme de séquence qui suit expose le fonctionnement du composant :

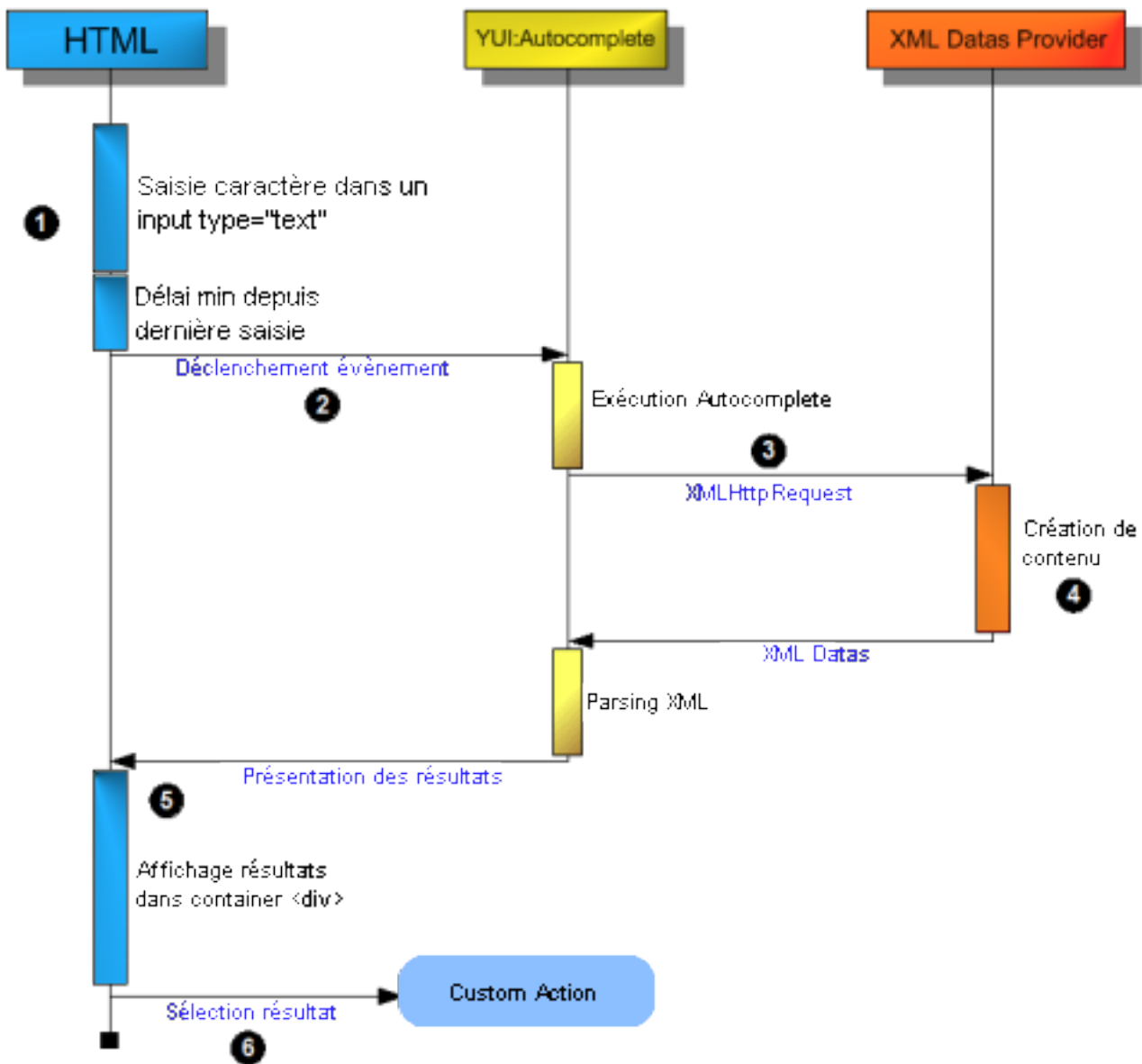


Diagramme de séquence

III - Mise en oeuvre du composant XSL Autocomplete

Le composant XSL est écrit dans un fichier nommé **yui.xsl** (qui a vocation à intégrer d'autres composants de la YUI Lib).

L'utilisation du composant XSL Autocomplete dans un XMLService requiert 4 étapes :

- 1 La déclaration du namespace yui dans le XSL et l'inclusion de la feuille de style du composant : yui.xsl
- 2 L'ajout des headers javascript.
- 3 L'insertion à l'endroit désiré du composant XSL (`<xsl:call-template name="yui:Autocomplete">`).
- 4 L'écriture du XMLService qui va produire les données de réponse à la saisie.

III-A - Déclaration du namespace yui

Nous déclarons le namespace yui en tête de fichier xsl : `xmlns:yui="http://developer.yahoo.com/yui/"`

Puis nous importons la librairie `yui.xsl` dans une WebForm :

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:msxsl="urn:schemas-microsoft-com:xslt" xmlns:xslc="http://xslcomponents.org/TR/WD-xslc"
xmlns:yui="http://developer.yahoo.com/yui/">
  <xsl:import href="../../../xslc.xsl"/>
  <xsl:import href="../../../Common.xsl"/>
  <xsl:import href="../../../yui.xsl"/>
```

yui.xsl hors cadre XMLRAD ?

Ceux qui souhaiteraient utiliser `yui.xsl` hors cadre XMLRAD, ne sont pas obligés d'importer `xslc.xsl` et `Common.xsl` qui sont des librairies XMLRAD.

De la même manière la déclaration des namespaces en tête de feuille de style n'impose strictement que `xmlns:yui`. Le reste est intrinsèquement lié à votre application.

Aussi, un seul répertoire virtuel est utilisé pour les fichiers javascripts nécessaires : [/SharedPortal/js](#).

Enfin si vous n'utilisez pas un XMLService pour les données de réponse à l'autocomplétion, alors la contrainte de votre service personnalisé reste de fournir une réponse au format XML.

III-B - Ajout des headers javascript

Les fichiers javascript utilisés par le composant proviennent de la lib Yahoo :

- 1 yahoo.js
- 2 dom.js
- 3 event.js
- 4 connection
- 5 animation.js

6 autocomplete.js

La librairie yui.xsl offre un template qui permet d'inclure les fichiers javascripts nécessaires à l'exécution du composant Autocomplete.

Il suffit d'invoquer le template de cette façon :

```
<xsl:template match="document">
<xsl:call-template name="xslc:Page">
  <xsl:with-param name="Head">
    <xsl:call-template name="yui:AutoComplete.Head"/>
  </xsl:with-param>
</xsl:template>
```

Le template yui:AutoComplete.Head va générer les liens de type :

```
<script language="javascript" src="/SharedPortal/yui_2.2.0/yahoo/yahoo-min.js"
xmlns:yui="http://developer.yahoo.com/yui/"></script>
```

III-C - Insertion du composant XSL

Nous allons maintenant insérer le composant XSL à l'endroit désiré de la feuille de style. Nous choisirons de le placer en tête du paramètre Body `xsl:with-param name="Body"` (Template `xsl:Page`).

Portez votre attention sur l'invocation du template nommé : `<xsl:call-template name="yui:AutoComplete">`

```
<xsl:template match="document">
<xsl:call-template name="xslc:Page">
  <xsl:with-param name="Body">
    <xsl:call-template name="yui:AutoComplete">
      <xsl:with-param name="Name">InputSearchCountry</xsl:with-param>
      <xsl:with-param name="InputAttributes"><Attributes size="20"/></xsl:with-param>
      <xsl:with-param name="UseShadow">true</xsl:with-param>
      <xsl:with-param name="DataXMLServiceName">DataSearchCOUNTRY</xsl:with-param>
      <xsl:with-param name="ResultRecordName">COUNTRY</xsl:with-param>
      <xsl:with-param name="ResultFields">
        <Field>
          <Name>COUNTRY_NAME</Name>
        </Field>
        <Field>
          <Name>COUNTRY_ID</Name>
        </Field>
        <Field>
          <Name>COUNTRY_POPULATION</Name>
        </Field>
      </xsl:with-param>
      <xsl:with-param name="TemplateResult">
        <span style="color:blue"><FieldName>COUNTRY_NAME</FieldName></span>
        <span style="color:purple">&#160;( <FieldName>COUNTRY_POPULATION</FieldName> )</span>
      </xsl:with-param> <!-- param name="TemplateResult" -->
      ...
    </xsl:call-template>
    ...
  </xsl:with-param> <!-- Body -->
  ...
</xsl:call-template> <!-- xslc:Page -->
...
```

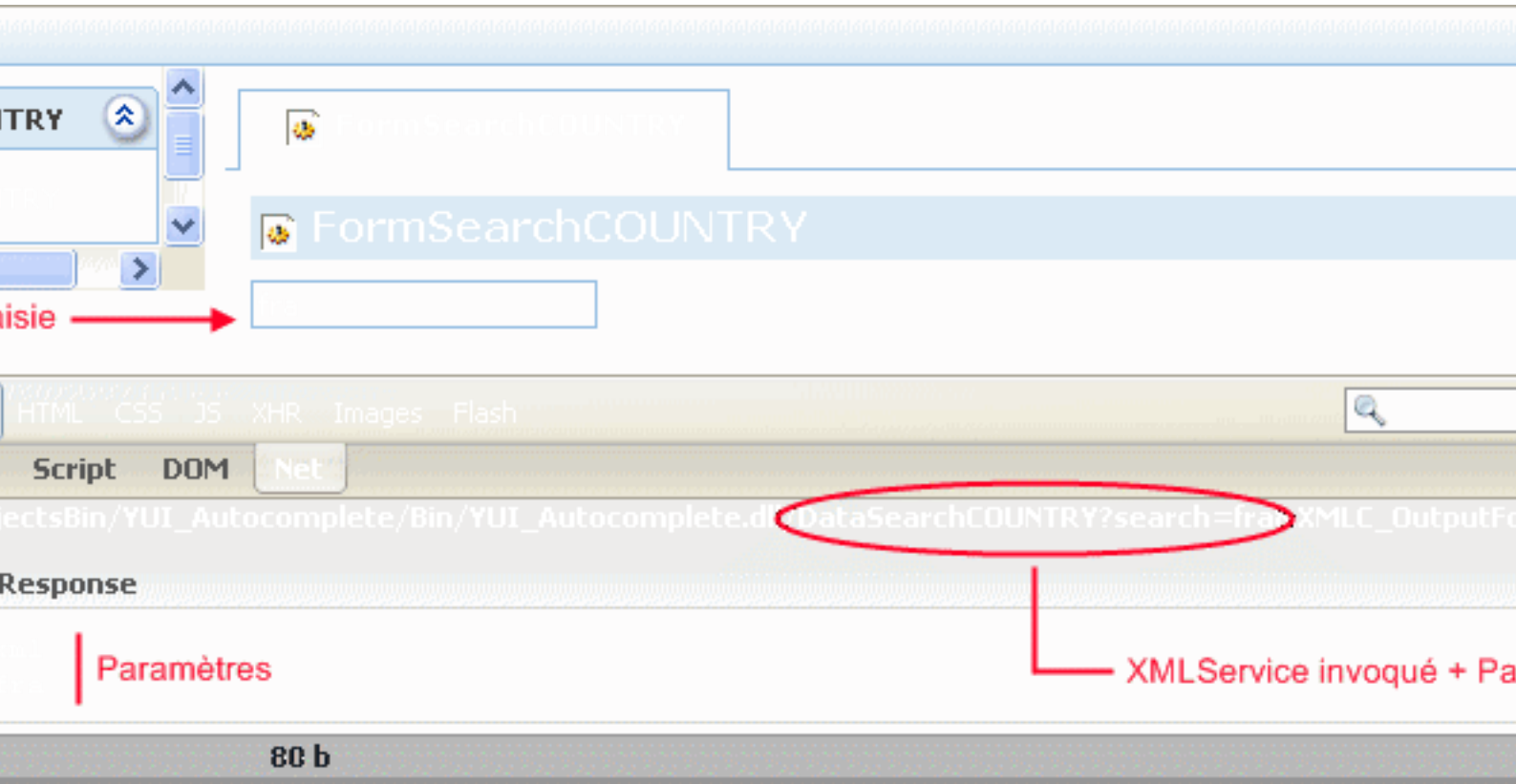
```
...  
</xsl:template>
```

Le template est appelé ici avec quelques paramètres :

Nom du paramètre	Description
Name	Nom qui sera donné au champ de saisie de l'autocomplete
InputAttributes	Attributs supplémentaires du champ de saisie comme la taille etc.
DatasXMLServiceName	Nom du XMLService qui va renvoyer les résultats de la recherche
ResultRecordName	Nom de chaque élément XML résultat
ResultFields	Nom des champs enfants de ResultRecordName qu'on utilisera à l'affichage
TemplateResult	Patron visuel html de présentation de chaque ligne résultat. Les champs dynamiques issus de la requête sont insérés à l'aide de la balise <FieldName>.

Ce template va insérer une zone de saisie (input type="text) dans le flux HTML.

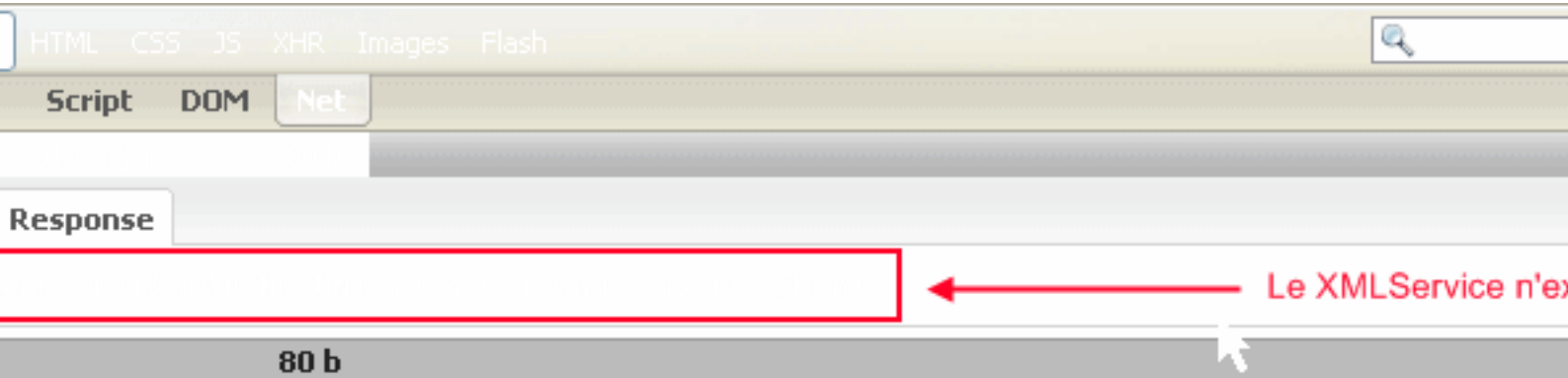
Dès que l'utilisateur va saisir quelques caractères, le navigateur va émettre une requête HTTP joignant notre XMLService, DataSearchCOUNTRY :



Le composant Autocomplete invoque le fournisseur de données

Pour rappel, XHR veut dire XMLHttpRequest. [Description Wikipedia](#)

Si on regarde le résultat retourné par la requête XHR :



Le XMLService DataSearchCOUNTRY n'existe pas encore !

Pour que le composant puisse afficher une liste de résultat, il faut que le XMLService responsable de la production des données existe et... fonctionne :-)

Nous allons donc créer ce service, qui va exécuter une requête SQL recherchant les états comportant le mot clé passé en paramètre. Rappelons que le paramètre s'appelle **search**.

III-D - Ajout du XMLService fournisseur de données

Le composant autocomplete va invoquer un service local évitant ainsi une requête XMLHttp "cross-domain". Ce service local pourra invoquer des services distants sur internet (WebServices, RSS, RDF...), récupérer des données dans une base de données voire dans un fichier XML etc...

Nous implémentons ce service sous la forme d'un XMLService.

Dans la section précédente III-C, nous avons appelé un template nommé avec un paramètre qui nous intéresse particulièrement :

```
<xsl:with-param name="DatasXMLServiceName">DataSearchCOUNTRY</xsl:with-param>
```

C'est le XMLService **DataSearchCOUNTRY** qui est responsable de la recherche / production des données qui seront affichées dans la liste déroulante de l'autocomplétion.

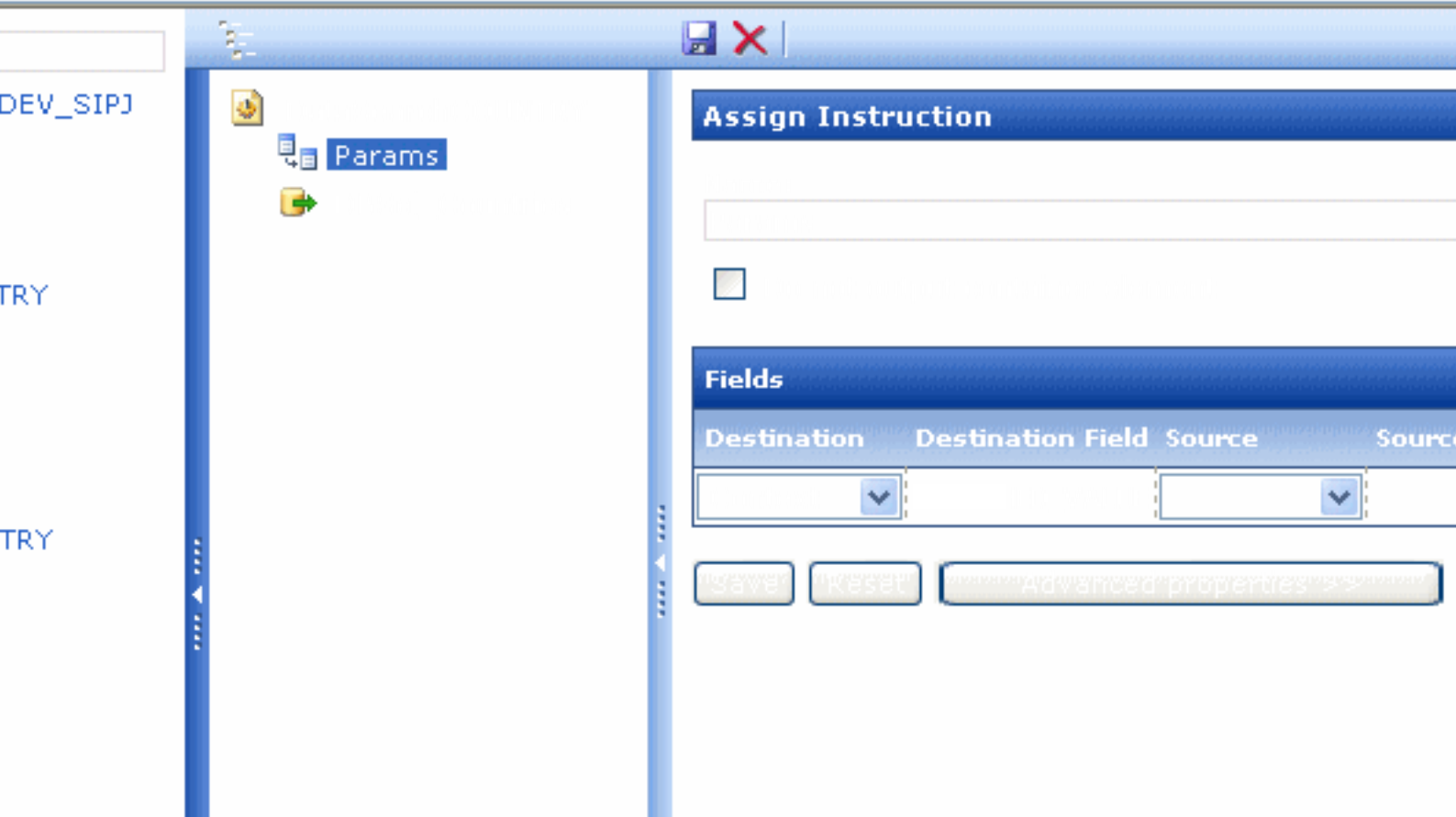
Parmi les paramètres HTTP que recevra DataSearchCOUNTRY, il y a la valeur qui a été saisie dans le champ input du formulaire : **q = "searched value"**

Le XMLGram se contente d'intégrer une XMLInstruction DBExtract : Les résultats seront retournés au format XML par le framework : Ce sont ces résultats que le composant Autocomplete va parser et présenter à l'utilisateur dans une liste déroulante.

Création du XMLService

Nous créons un XMLService de type "XMLGram Only" nommé "DataSearchCOUNTRY".

Dans le XMLGram, nous débutons par une XMLInstruction **Assign** qui va récupérer la valeur du paramètre **search** en l'entourant du caractère %. La donnée SEARCHED_VALUE représentera la valeur à rechercher. Cela nous servira avec la condition SQL LIKE :



Caractère pourcent ajouté au mot clé recherché

Puis nous ajoutons une XMLInstruction **DBExtract** qui va rechercher les pays dont le nom comporte le mot clé recherché :

The screenshot shows the XMLRAD configuration interface for a DBExtract instruction. On the left, a tree view shows a project named 'DBSel_Countries'. The main configuration area is divided into several sections:

- DBExtract Instruction:** Contains several input fields, dropdown menus, and a checked checkbox.
- Fields:** A table with columns 'Name', 'Source', 'Type', and 'Extract'. The 'Name' column is currently empty.
- Params:** A table with columns 'Name', 'Type', 'Value', and 'Comments'. The 'Name' column is circled in red.
- SQL statement:** A text area for the SQL query, which is also circled in red.

XMLInstruction DBExtract, recherche SQL des pays

Nous essayons à nouveau notre composant, en saisissant "af" dans la zone de saisie.

Firebug nous indique qu'une nouvelle requête XHR a été exécutée. Notre XMLService DataSearchCOUNTRY a été invoqué. Allons voir la réponse retournée par l'application XMLRAD :

Response

```
<COUNTRY><COUNTRY_ID>-465</COUNTRY_ID><COUNTRY_NAME>Afghanistan</COUNTRY_NAME><COUNTRY_POPULATION>28700000</COUNTRY_POPULATION></COUNTRY>
```

380 b

Réponse XML

Nous constatons que le flux XML retourné a été généré automatiquement par le framework XMLCLX.

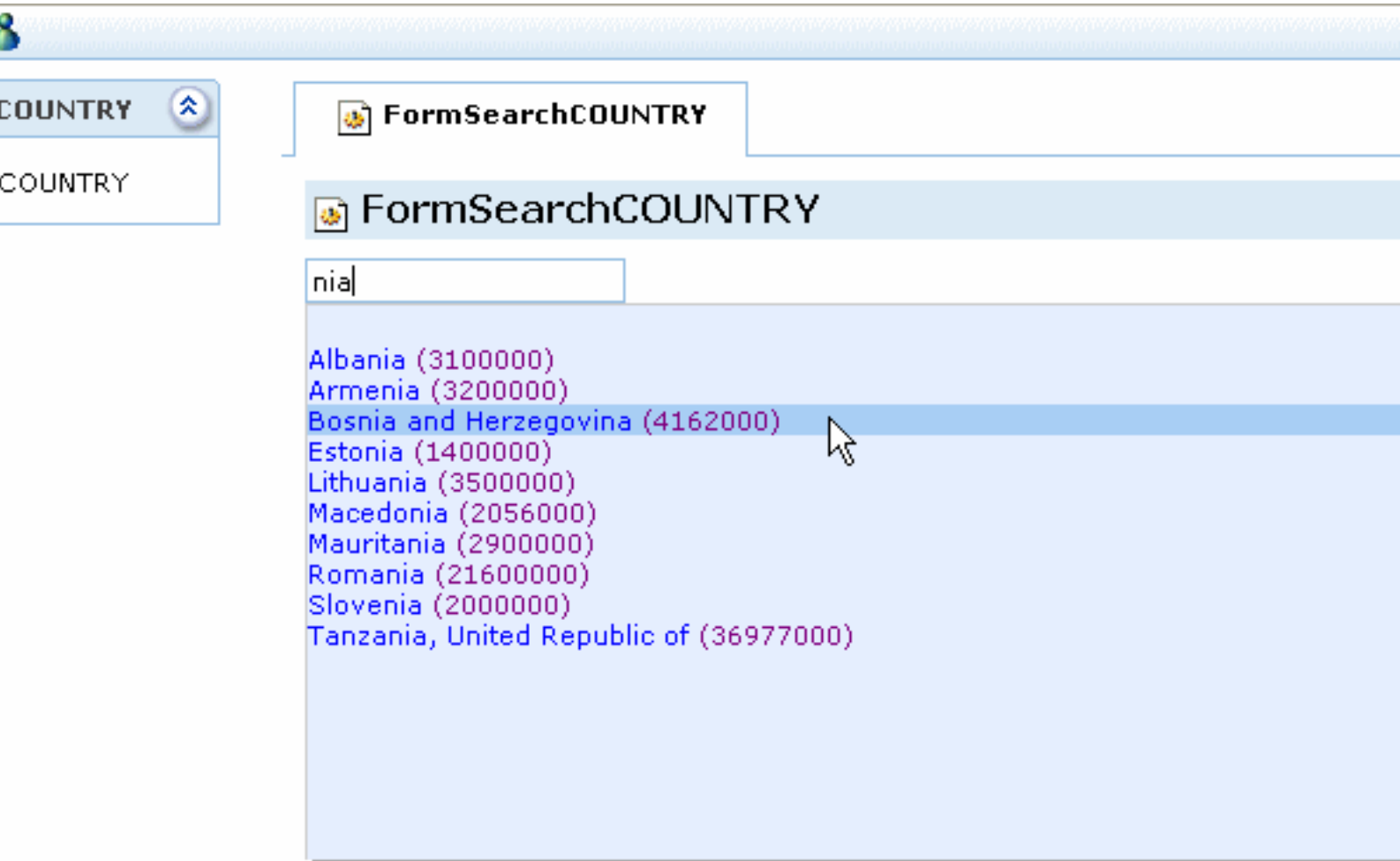
Nous avons 3 résultats, chacun présenté dans une balise **<COUNTRY>**.

C'est à ce moment que le composant Autocomplete va parser les réponses et présenter les résultats en suivant le template visuel que nous avons indiqué dans le paramètre **TemplateResult** :

```
<xsl:with-param name="TemplateResult">
  <span style="color:blue"><FieldName>COUNTRY_NAME</FieldName></span>
```

```
<span style="color:purple">&#160;(<FieldName>COUNTRY_POPULATION</FieldName>)</span>  
</xsl:with-param>
```

Ca donne le résultat suivant :



The screenshot shows a web application interface. On the left, there is a sidebar with a 'COUNTRY' menu. The main content area is titled 'FormSearchCOUNTRY'. Below the title, there is an input field containing the text 'nia'. A dropdown menu is open, displaying a list of countries and their populations. The list includes: Albania (3100000), Armenia (3200000), Bosnia and Herzegovina (4162000), Estonia (1400000), Lithuania (3500000), Macedonia (2056000), Mauritania (2900000), Romania (21600000), Slovenia (2000000), and Tanzania, United Republic of (36977000). The 'Bosnia and Herzegovina' entry is highlighted with a blue background, and a mouse cursor is pointing at it.

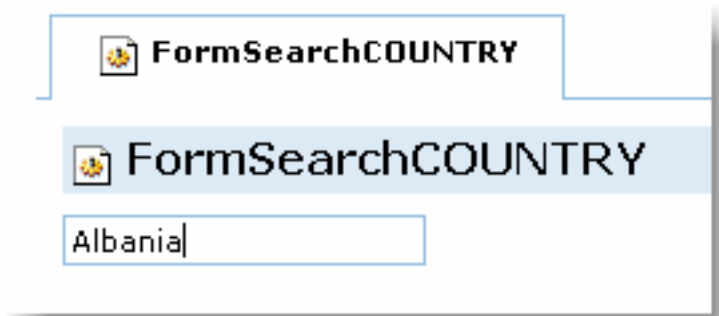
Présentation des résultats

Si on clique sur un des résultats, par exemple "Albania (3100000)", seul le mot "Albania" s'affichera dans la zone de saisie.

Pourquoi ?

Parce que dans le paramètre <ResultFields> du composant XSL, le premier élément <Field> était COUNTRY_NAME. Si vous aviez voulu afficher la population, alors il aurait fallu placer en premier le champ Field dont l'élément NAME est COUNTRY_POPULATION.

Nous verrons cependant plus loin qu'il est possible d'afficher la valeur d'un ou plusieurs champs aux endroits souhaités.



IV - Paramètres du composant

IV-A - Présentation

Nous allons aller un peu plus loin, car notre composant recèle encore quelques surprises... Et puis surtout, c'est bien drôle d'afficher du texte dans un champ de saisie suite à une sélection combien même bien présentée, mais dans la vraie vie on a souvent des besoins plus sournois !

Par exemple, comment faire pour remplir un ID caché dans un champ hidden lors de la sélection ? Aussi, comment faire pour rechercher des données sur un site externe (RDF, XML, RSS etc.) ? Ou alors, comment permettre une saisie dans un champ de type area et non plus un input text, de façon à rechercher successivement des adresses email ?

Des exemples, des exemples etc.

IV-B - Référence paramètres

Dans le tableau suivant, sont listés la liste des paramètres de la version courante de XSL:Autocomplete :

Nom du paramètre	Description
Name	Nom qui sera donné au champ de saisie de l'autocomplete
InputAttributes	Attributs supplémentaires du champ de saisie comme la taille etc.
InputType	2 valeurs possibles : input (par défaut) ou area. La saisie s'effectuera respectivement dans une zone input type="text" ou type="area"
UseShadow	Affiche ou non une ombre autour des résultats proposés
DelimChar	Caractère séparant le texte affiché et le dernier texte à rechercher, si plusieurs textes se succèdent dans la zone de saisie
DatasXMLServiceName	Nom du XMLService qui va renvoyer les résultats de la recherche
QueryParamName	Nom du paramètre http qui contiendra la valeur à rechercher, "search" par défaut.
QueryDelay	Délai après la dernière saisie de caractères avant exécution de la requête fournisseur de données. (0.5 sec par défaut).
CustomParam	Si la requête XHR doit être accompagnée de paramètres constants
ResultRecordName	Nom de chaque élément XML résultat
ResultFields	Nom des champs enfants de ResultRecordName qu'on utilisera à l'affichage
TemplateResult	Patron visuel html de présentation de chaque ligne résultat. Les champs dynamiques issus de la requête sont insérés à l'aide de la balise <FieldName>.

MaxResultsDisplayed	Nombre maximal de résultats proposés. 100 par défaut.
ActionOnKeyDown	Javascript à exécuter lors d'un keydown dans la zone de saisie
ActionOnSelect	Javascript à exécuter lorsqu'un résultat est sélectionné
ActionOnDataReturn	Javascript à exécuter lorsque les données de recherche sont renvoyées par le proxy fournisseur de données
ActionOnSelect	Javascript à exécuter lorsqu'un résultat est sélectionné

IV-C - Exemple : Mise à jour de champs lors de la sélection

Imaginez que vous êtes en train de remplir un formulaire concernant une personne. Vous devez indiquer son pays d'appartenance et pour ce faire, vous décidez de mettre en place un autocomplete.

Lorsque l'utilisateur sélectionnera un pays proposé, le composant devra mettre à jour l'identifiant (Pays_ID) dans un champ caché du formulaire qui provoquera l'insertion ou la mise à jour de la personne.

Il faut déjà prévoir le formulaire :

```
<form name="MainForm" method="POST" action="{/document/Aliases/YUI_AutocompleteDLL}InsertPERSON">
  <input type="hidden" name="Pays_ID" id="Pays_ID"/>
  ... <!-- autres champs du formulaire -->
```

Maintenant, nous allons indiquer dans les paramètres du template que le champ Pays_ID doit être mis à jour lors de la sélection (avec la valeur COUNTRY_ID de l'enregistrement respectif).

Vous vous rappelez quand on a énuméré les champs utiles pour le composant Autocomplete, dans le paramètre `<ResultFields><Fields>...</Fields></ResultFields>` ?

Nous allons faire la même chose avec une légère addition : l'élément **SetDomID** :

```
<xsl:with-param name="ResultFields">
  <Field>
    <Name>COUNTRY_NAME</Name>
  </Field>
  <Field>
    <Name>COUNTRY_ID</Name>
    <SetDomID>Pays_ID</SetDomID>
  </Field>
  <Field>
    <Name>COUNTRY_POPULATION</Name>
    <SetDomID>Pays_Population</SetDomID>
  </Field>
</xsl:with-param>
```

Maintenant assurons nous du bon fonctionnement en surveillant la valeur du champ de formulaire **Pays_ID** quand on sélectionnera un pays.

Pendant qu'on y est, nous afficherons au passage la population du pays sélectionné, dans une balise `` identifiée par : id = "Pays_Population".

Juste avant la sélection :

New PERSON

Albania (3100000)
Armenia (3200000) ← Sélection
Bosnia and Herzegovina (.162000)
Estonia (1400000)
Lithuania (3500000)
Macedonia (2056000)
Mauritania (2900000)

```
span.cFieldCaption < td < tr < tbody < table < form < td < tr < tbody < table.cITal
```

```
ML CSS Script DOM Net Options Style Lay
```

```
s="PageTitle">
```

```
kslcApplicationMessages" style="display:
```

```
ns:yui="http://developer.yahoo.com/yui/" action="/ProjectsBin/YUI Autoc
```

```
id="Pays_ID" type="hidden" name="Pays_ID"/>
```

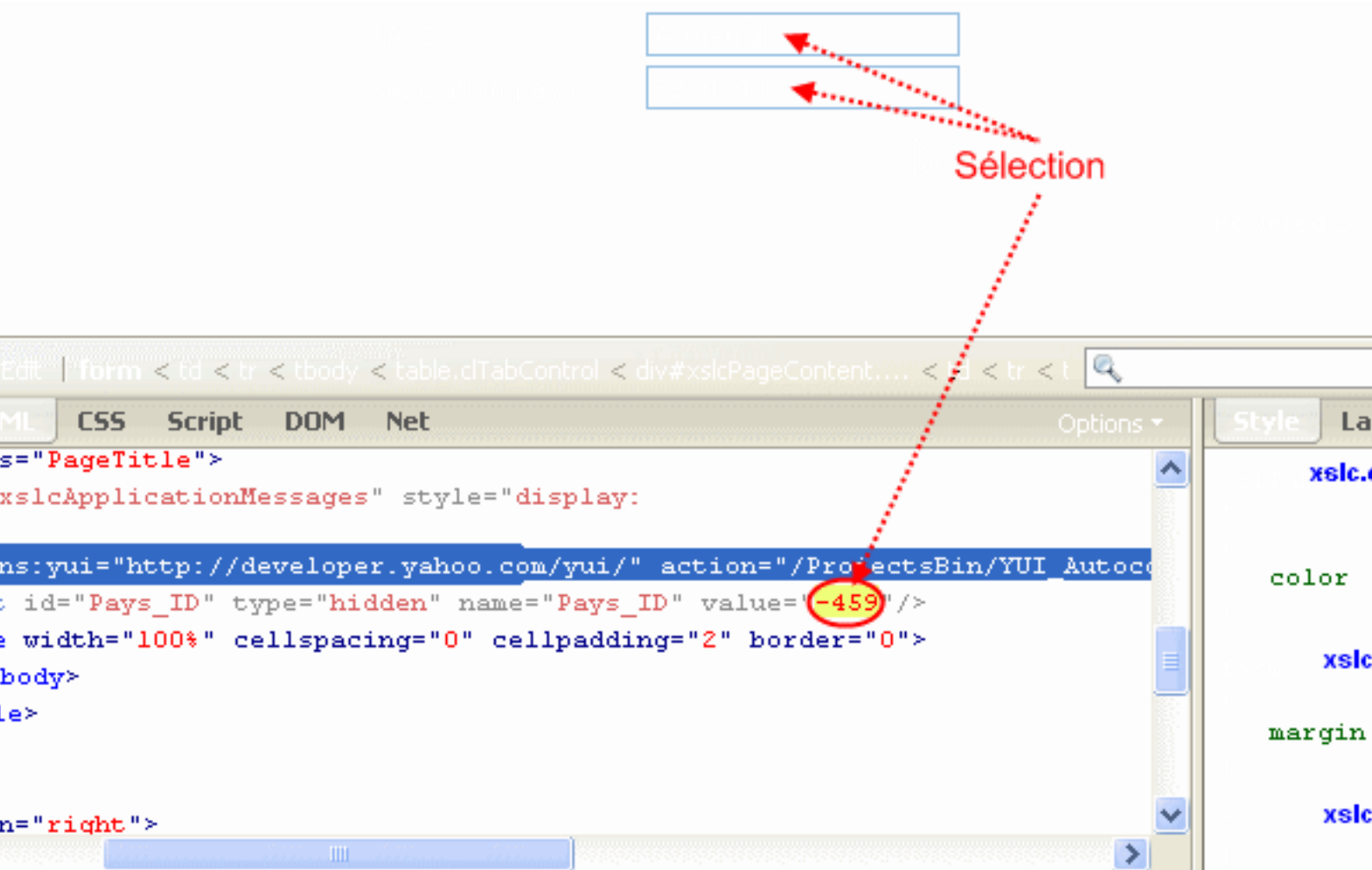
```
width="100%" cellpadding="2" border="0">
```

```
tbody>
```

```
e>
```

```
n="right">
```

Juste après la sélection :



*Très pratique. Malgré tout, une des limites actuelles du composant est qu'il ne sait mettre à jour **que des attributs "value" sur un élément identifiable du DOM.***

Par exemple, il n'est actuellement pas possible de remplir le contenu d'une balise ``, car il faudrait dans ce cas modifier la valeur de l'élément enfant `innerHTML` (ou technique équivalente). C'est à l'étude pour la prochaine version du composant.

```
<xsl:with-param name="InputType">textarea</with-param><xsl:with-param name="InputAttributes">
  <Attributes size="20" style="display:none"/>
</xsl:with-param>
```

On peut faire en sorte que la zone d'input soit une text area (pour une liste d'emails par exemple).

On peut également agir sur les attributs de cette zone d'input pour modifier sa taille, couleur, visibilité etc.

IV-D - Exemple : Saisie multiple, choix de destinataires email

Passons à un autre exemple : Façon GMail, nous allons rechercher et sélectionner successivement plusieurs destinataires email.

Nous allons avoir besoin d'une surface de saisie plus grande (un textarea...), et que la recherche s'effectue sur une partie du contenu, en l'occurrence le dernier morceau saisi.

Allons-y ! Insérons le code suivant dans un nouveau XMLService, que nous nommerons FormSearchEMAIL :

```
<xsl:call-template name="yui:AutoComplete">
  <xsl:with-param name="Name">search</xsl:with-param>
  <xsl:with-param name="InputType">textarea</xsl:with-param>
  <xsl:with-param name="InputAttributes">
    <Attributes cols="100" rows="3"/>
  </xsl:with-param>
  <xsl:with-param name="UseShadow">>true</xsl:with-param>
  <xsl:with-param name="DelimChar">\n</xsl:with-param>
  <xsl:with-param name="DatasXMLServiceName">DataSearchEMAIL</xsl:with-param>
  <xsl:with-param name="ResultRecordName">EMAIL</xsl:with-param>
  <xsl:with-param name="ResultFields">
    <Field>
      <Name>PSN_EMAIL</Name>
    </Field>
    <Field>
      <Name>PSN_FIRST_NAME</Name>
    </Field>
    <Field>
      <Name>PSN_LAST_NAME</Name>
    </Field>
  </xsl:with-param>
  <xsl:with-param name="TemplateResult">
    <div>
      
      <span style="color:#376A94;"><FieldName>PSN_EMAIL</FieldName></span>
      <br/>
      <FieldName>PSN_FIRST_NAME</FieldName>&#160;<FieldName>PSN_LAST_NAME</FieldName>
    </div>
    <div style="height:4px;">&#160;</div>
  </xsl:with-param>
</xsl:call-template>
```

De "nouveaux" paramètres font leur apparition :

Paramètre InputType :

```
<xsl:with-param name="InputType">textarea</xsl:with-param>
```

La saisie s'effectuera dans un champ de type textarea !

Paramètre InputAttributes :

```
<xsl:with-param name="InputAttributes">
  <Attributes cols="100" rows="3"/>
</xsl:with-param>
```

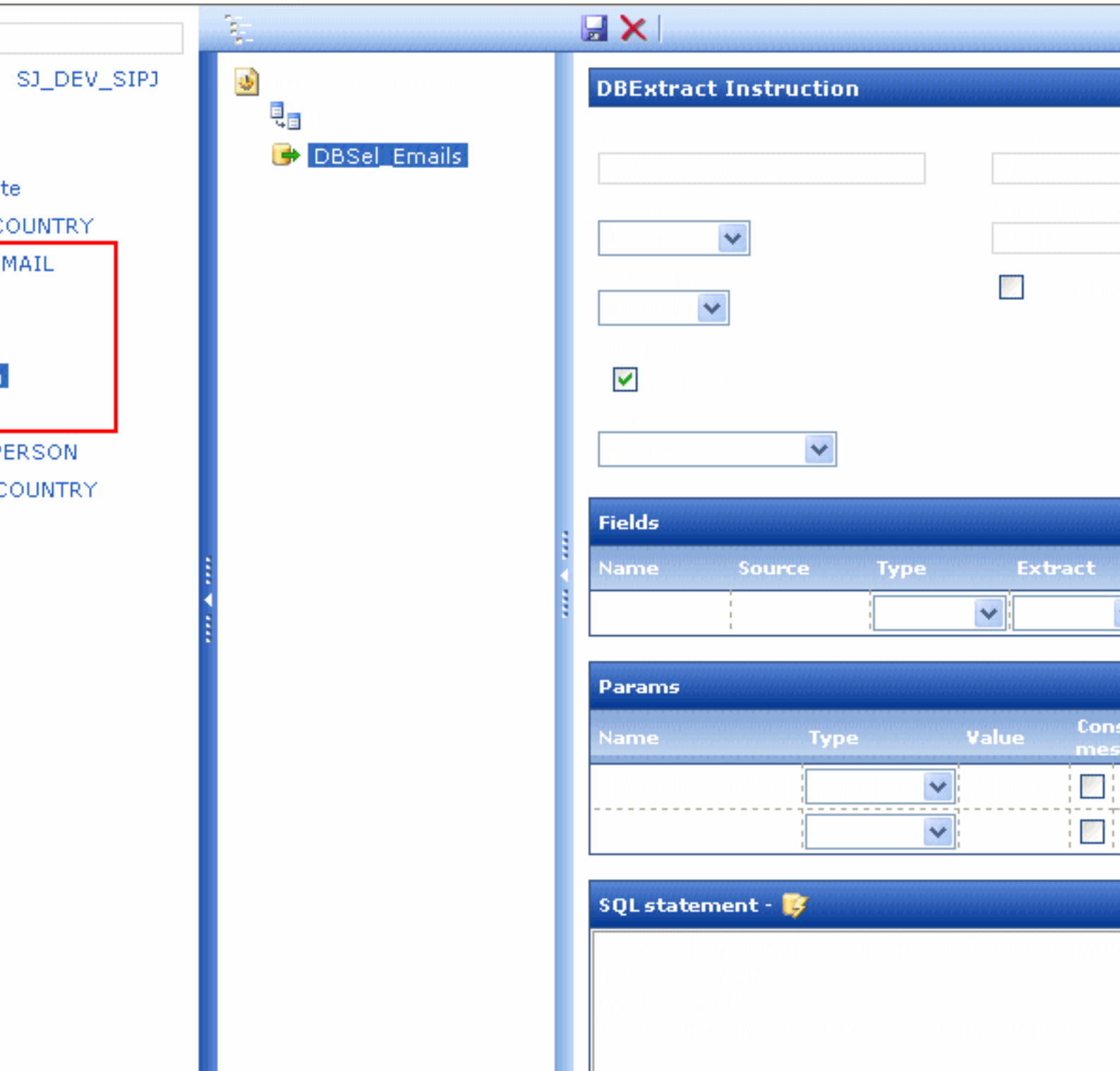
Les attributs de ce champ textarea : 100 colonnes et 3 lignes.

Paramètre **DelimChar** :


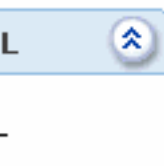

```
<xsl:with-param name="DelimChar">\n</xsl:with-param>
```

Le paramètre DelimChar permet d'indiquer que la recherche des emails **s'effectuera sur le texte qui suit le dernier saut de ligne (\n)**

Nous créons le XMLService DataSearchEMAIL qui fournira les réponses pour les emails trouvés dans la table PERSON :











Le résultat en image montre combien il a été aisé d'implémenter un bloc de sélection de destinataires.

 **FormSearchEMAIL** **FormSearchEMAIL**

Sélectionner des emails :

jvollet@fr.delos.us
nmussat@fr.delos.us
jpbempel@fr.delos.us
delos



-  kee.wee@e-delos.com
Kee Wee
-  kdean@e-delos.com
Kevin Dean
-  jonathan.colomb@e-delos.com
Jonathan Colomb
-  ibecerril@fr.delos.us
Ivan Becerril
-  jvollet@fr.delos.us
Jerome Vollet
-  gbreselec@fr.delos.us
Gilles Breselec
-  jpbempel@fr.delos.us
Jean-Philippe Bempel
-  areala@fr.delos.us

Dans un de mes projets, au lieu de rechercher les emails dans une base de données, notre proxy recherchait dans un annuaire LDAP... La mécanique côté client est exactement la même. Le proxy qui recherchait dans l'annuaire LDAP se chargeait seulement de formater les résultats au format XML, chaque résultat étant présenté dans une balise **<EMAIL>**.

IV-E - Exemple : Recherche et vérification d'adresse via la géolocalisation de GoogleMaps

Un dernier petit exemple pour la route : Cette fois, la recherche s'effectuera sur un domaine extérieur à l'application.

Comme nous ne pouvons pas accéder directement à un domaine externe avec une requête XHR, nous allons écrire un proxy applicatif local. Ce dernier invoquera lui-même l'API GoogleMaps.

Décrivons succinctement les étapes :


- 1 L'utilisateur saisit un morceau d'adresse dans un champ de saisie

- 2 Le composant Autocomplete envoie une requête vers un XMLService local à l'application (XHR locale !)
- 3 Ce dernier XMLService invoque en HTTP l'API Google Maps et produit un résultat XML
- 4 Le composant Autocomplete présente les résultats
- 5 L'utilisateur sélectionne les résultats, et les champs correspondant à l'adresse s'autoalimentent



Pour rappel, il est obligatoire que la requête XHR s'effectue sur le domaine local de l'application.

C'est pour cela que la plupart des sites mettent en oeuvre cette technique de proxy local.

*Il semblerait que les choses changent à l'avenir :  **La fin de l'interdiction d'invocation XMLHttpRequest cross-sites ?***

Nous commencerons par écrire le proxy local sous la forme d'un XMLService de type "XMLGram Only". Puis nous intégrerons et paramètrons le composant Autocomplete dans la forme de recherche.

Ecriture du proxy local DataSearchADRESSE :

Ce service doit interroger l'API Google Maps de géolocalisation. Pour ce faire, il faut invoquer l'url <http://maps.google.com/maps/geo> en passant en paramètre l'**adresse recherchée**, le **paramètre output = xml** et la **clé GoogleMaps** :

google.com/maps/geo?q=8+route+des+gardes&output=xml&key=ABQIAAAAHTRUQSC18CgbO81qqFodmBQ

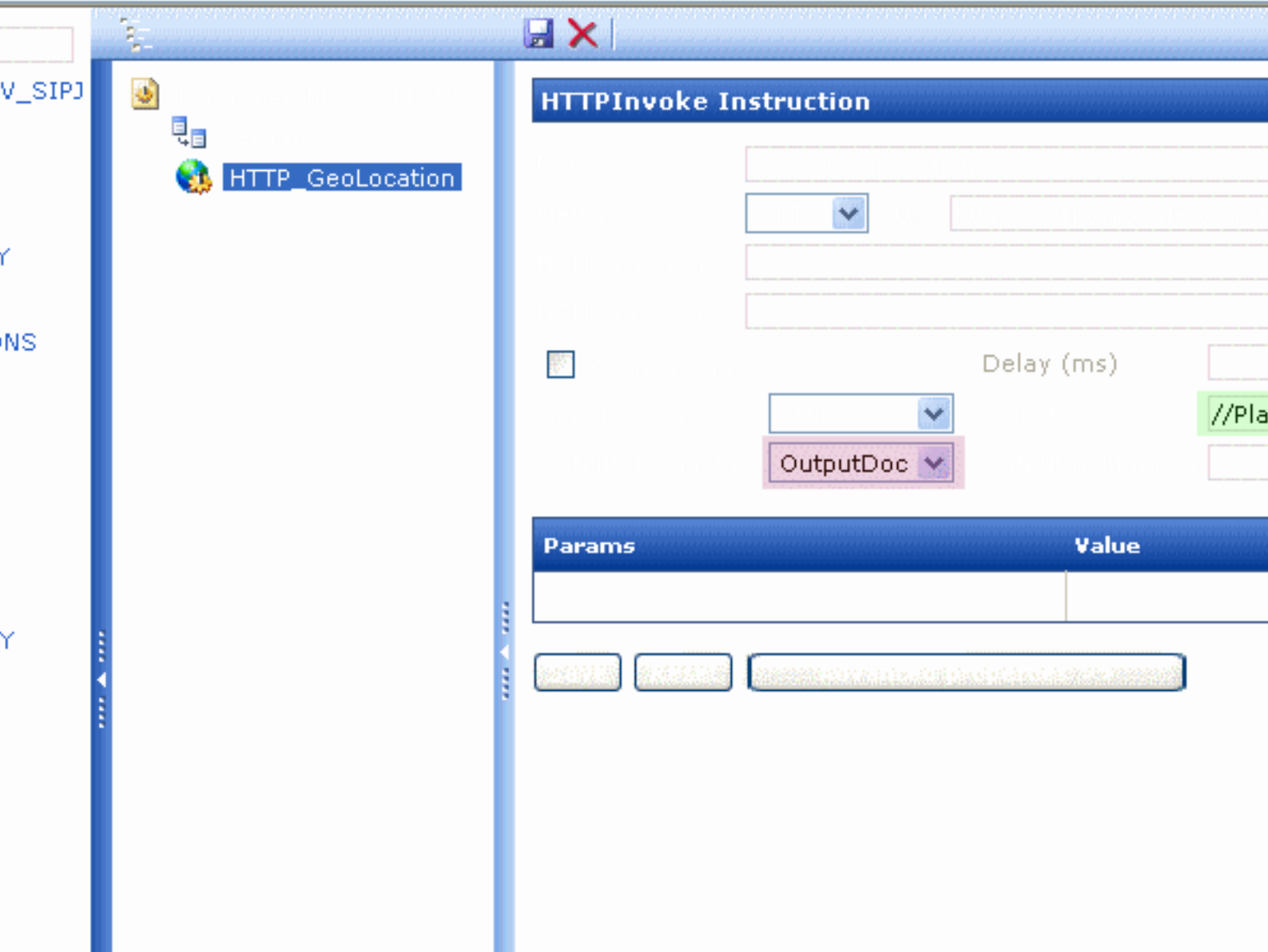
```

on="1.0" encoding="UTF-8" ?>
"http://earth.google.com/kml/2.0">
e>
</name>
>
mark id=" ">
mark id=" ">
mark id=" ">
ess>
</address>
ressDetails Accuracy=" " xmlns="urn:oasis:names:tc:ciq:xdschema:xAL:2.0">
untry>
CountryNameCode> </CountryNameCode>
AdministrativeArea>
<AdministrativeAreaName> </AdministrativeAreaName>
<SubAdministrativeArea>
<SubAdministrativeAreaName>
</SubAdministrativeAreaName>
- <Locality>
<LocalityName> </LocalityName>
- <Thoroughfare>
<ThoroughfareName> </ThoroughfareName>
</Thoroughfare>

```

Le résultat XML renvoyé par Google Maps

Nous créons le XMLGram avec une XMLInstruction [HttpInvoke](#) qui se charge d'interroger Google Maps :



HttpInvoke de Google Maps

Cette instruction HttpInvoke va parser le flux XML retourné par Google Maps, et sélectionner les éléments **<Placemark>**. Pour chaque élément **<Placemark>** sera généré une "ligne" de réponse dans le container visuel de l'autocomplete.

Il nous reste à intégrer le composant Autocomplete dans une form, en lui indiquant de rechercher une adresse via le XMLService DataSearchLOCATIONS.

Création du XMLService

A cet effet, nous créons un nouveau XMLService, que nous nommons **FormGEOLOC**.

Nous insérons notre composant XSL, paramétré pour l'occasion.

Voici l'extrait de code XSL :

```
<xsl:call-template name="yui:AutoComplete">
  <xsl:with-param name="Name">InputSearchLOCATIONS</xsl:with-param>
  <xsl:with-param name="InputAttributes">
    <Attributes size="70"/>
  </xsl:with-param>
  <xsl:with-param name="UseShadow">>true</xsl:with-param>
  <xsl:with-param name="DatasXMLServiceName">DataSearchLOCATIONS</xsl:with-param>
  <xsl:with-param name="QueryParamName">q</xsl:with-param>
  <xsl:with-param name="QueryDelay">4</xsl:with-param>
  <xsl:with-param name="ResultRecordName">Placemark</xsl:with-param>
  <xsl:with-param name="ResultFields">
    <Field>
      <Name>address</Name>
    </Field>
    <Field>
      <Name>coordinates</Name>
    </Field>
    <Field>
      <Name>ThoroughfareName</Name>
      <SetDomID>ADRESSE_AFF</SetDomID>
    </Field>
    <Field>
      <Name>LocalityName</Name>
      <SetDomID>VILLE_AFF</SetDomID>
    </Field>
    <Field>
      <Name>PostalCodeNumber</Name>
      <SetDomID>CP_AFF</SetDomID>
    </Field>
  </xsl:with-param>
  <xsl:with-param name="TemplateResult">
    <table border="0" cellpadding="1" cellspacing="0" width="100%"
style="margin-bottom:2px;">
      <tr>
        <td></td>
        <td>
          <span style="color:#376A94;">
            <FieldName>address</FieldName>
          </span>
        </td>
        <td>
          <span style="color:#FF567A;">
            <FieldName>coordinates</FieldName>
          </span>
        </td>
      </tr>
    </table>
  </xsl:with-param>
</xsl:call-template>
```

Des "nouveaux" paramètres font leur apparition :

Paramètre **QueryDelay** :

```
<xsl:with-param name="QueryDelay">4</xsl:with-param>
```

Il s'agit d'augmenter le délai d'attente avant de lancer une recherche, car la saisie d'une adresse prend un peu de temps :-)


Nous portons cette valeur à 4 secondes.

Paramètre **QueryParamName** :

```
<xsl:with-param name="QueryParamName">q</xsl:with-param>
```

Par défaut, c'est le paramètre **search** qui contient la valeur recherchée et transmis avec la requête HTTP.

Nous pouvons changer le nom de ce paramètre, comme ici, où nous l'avons renommé "q".

 Notez que nous retrouvons nos paramètres **<SetDomID>**.

Dès que nous aurons sélectionné un résultat, les champs de saisie détaillant l'adresse (Adresse, CP, Ville) seront remplis automatiquement.

Nous pouvons découvrir le résultat :

 **FormGeoloc**

FormGeoloc


Rechercher :

	Rue des Saussaies, 78510 Triel-sur-Seine, France	2.019385,48.979329,0
	Rue des Saussaies, 94230 Cachan, France	2.337246,48.786798,0
	Rue des Saussaies, 91590 Baulne, France	2.368670,48.507432,0
	Rue des Saussaies, 21500 Quincy-le-Vicomte, France	4.255444,47.607563,0
	Rue des Saussaies, 51700 Verneuil, France	3.675503,49.092938,0
	Rue des Saussaies, 75008 8ème Arrondissement, Paris, France	2.317804,48.871364,0
	Rue des Saussaies, 02000 Laon, France	3.635958,49.566077,0
	Rue des Saussaies, 94190 Villeneuve-Saint-Georges, France	2.458200,48.735219,0
	Rue des Saussaies, 63720 Entraigues, France	3.264006,45.888783,0
	Rue des Saussaies, 51500 Taissy, France	4.093274,49.212165,0

Recherche d'une adresse

Et quand on sélectionne une adresse, nos champs de saisie sont bien remplis dans la foulée.

Merci Google. Et... merci Yahoo ! (D'ailleurs ils auraient sans doute préféré que je passe par leur API à eux ! C'est hors sujet, mais il faut reconnaître que l'API Google Map présente à mon sens les qualités d'une meilleure intégration dans nos applications web au regard de ses concurrentes (Yahoo, ViaMichelin etc.). Mais c'est un autre débat...

 **FormGeoloc** **FormGeoloc**Rechercher : Adresse CP Ville

Nous avons sélectionné une adresse...



V - Conclusion

V-A - Générales

Vous avez pu constater que la mise en oeuvre de l'autocomplete en partant de ZERO pouvait rebuter plus d'un développeur, même bien documenté sur le site de Yahoo.

Alors, une encapsulation telle que notre composant XSL Autocomplete nous facilite grandement la tâche. Il est vrai que du coup, on ne maîtrise plus trop le fond javascript qui est généré. Mais il est très simple d'aller vérifier dans le code source de la page HTML générée. Le code javascript relatif à l'autocomplete n'est pas obfusqué et vous comprendrez très vite le travail effectué par le composant, rien de bien obscur.

Il est temps de s'arrêter sur notre dernier exemple de Géolocalisation pour ce tutorial, j'en ai d'autres sur le bout de la langue, mais autant en garder un peu en exclu pour mes stagiaires (vous aussi vous pouvez le faire ! je veux dire... : Faire partie de mes stagiaires :-)), voire dans quelque temps une partie 2 de ce tuto ?

En effet on pourrait aborder des questions, comme : Comment prendre la main en javascript lorsqu'on sélectionne une réponse ? Comment afficher un indicateur visuel d'attente lorsqu'une recherche est en cours ? Comment afficher un bel icône dès que les résultats sont extraits ? etc. J'espère que cela vous aura donné des idées, pour apporter votre contribution voire créer d'autres composants XSL, et surtout apporter vos critiques et questions. Pensez aux forums  XMLRAD ou  XSL

V-B - Améliorations

Bien que je fournisse les sources avec ce tutoriel, le composant n'est pas encore relasé publiquement. Les raisons est que j'attends encore des remarques de nouveaux utilisateurs, afin de fixer définitivement la grammaire XML, et plus généralement "la philosophie" de la version 1.

Une autre raison est que pendant l'écriture de ce document, YAHOO a publié une nouvelle version de son composant Autocomplete avec des changements qui impactent le composant XSL. Il faut donc que je le modifie en conséquence.

Pas de panique, les sources livrées ici sont accompagnées de la version Yahoo qui lui permet de fonctionner.


Hors l'adaptation à la nouvelle version, les améliorations apportées le seront en fonction de vos remarques et contributions, ainsi des des todos en attente.


Par exemple, j'en parlais plus haut, il serait bien que `<SetDomID>` permettre d'affecter sur sélection, une valeur à tout type de balise, et pas seulement aux balises qui comportent un attribut "value".

Enfin, il reste à externaliser les css pour permettre un skinnage personnalisé.

V-C - Et les sources... ?

Sur un plateau :

	Description
	La librairie Yahoo à décompresser dans \SharedPortal\js

	Les sources du projet XMLRAD 2006R1 (IIS / JScript)
---	---

