Improve the usability of Web applications with type-ahead input fields using JSF, AJAX, and Web services in Rational Application Developer V7

Yury Kats

Type-ahead input fields (also known as suggest or autocomplete) are gaining popularity on the Web. Fields of this kind are programmed to guess what the user is typing and provide suggestions for the user to choose from. Rational Application Developer version 7.0 provides JSF components and AJAX functionality that make creating such fields fast and easy. This article explains type-ahead support in Rational Application Developer V7 and walks you through the process of creating two type-ahead fields -- one that uses a local data source to generate suggestions and one that uses a publicly available Web service.

How type-ahead works

The purpose of the type-ahead input fields is to improve the user experience. When someone is entering something in an input field on a page, it is often possible for the computer to make guesses about what that person is going to type, and thus complete the data entry for the user.

For example, if you the user is required to enter the name of the state he lives in, and he starts entering Mass, it is very likely that he is about to type Massachusetts. In this case, the computer could automatically complete the word by addingachusetts to the end of Mass, which the user already entered. This saves time and reduces the number of potential errors, because the user has less likelihood of making a spelling mistake.

Although it is often possible to correctly guess what the user intends to type next, this isn't always the case. What if, instead of typing Mass, the user entered just Ma? There are three states that
begin with *Ma*; therefore, it's not possible to predict which of the three the user is going to type. In this case, she is offered a list of choices to pick from: Maine, Maryland, and Massachusetts. The choices are usually presented in a list box under the input field. When she picks one of the choices offered, that choice is copied into the input field, thereby completing the entry. However, if she keeps typing, the number of suggestions offered decreases, because the more data that users enter, the better the computer software can guess what results they expect.

The technology that makes such fields possible in a Web browser is fairly simple, but it hasn't been widely used until recently. For the Web browser to get the list of suggestions, it needs to communicate with the server in a way that the user does not notice. This background communication between the browser and the server is commonly called AJAX (an abbreviation for asynchronous JavaScript™ and XML), and it's becoming very popular on various Web sites.

The idea behind a type-ahead field is very simple: As soon as the user enters a character into the input field, the browser issues an AJAX request to the server by passing the current content of that field as the parameter. The server evaluates the content and returns a list of suggestions to the browser. The browser then displays a list box next to the input field that offers the suggestions from the server. If the user picks one of the suggestions, it is copied into the input field.

There are four major parts required to implement this service:

- The browser needs JavaScript that creates and processes AJAX requests for the background communication with the server.
- The browser also needs JavaScript that defines how to present the list of suggestions to the user when it is received from the server.
- The server needs code to instruct it to receive AJAX requests from the browser and respond to them.
- The server also needs code for guessing what the user is intending to type or expecting and for generating suggestions for the user to choose from.

Although the idea is simple, the implementation of all of these parts can be very complex. Luckily, the JavaServer™ Faces (JSF) library in Rational Application Developer V7 takes care of the first three parts, completely relieving you of the need to write client-side JavaScript and client-server AJAX communication. After you add the type-ahead JSF component to the page, all you need to do is to provide server-side code to generate suggestions (the fourth of the four requirements). In this article, you'll learn two ways of doing that: writing your own code to use a local data source or using a Web service.

**Initial setup**

First, you need to create a Web project, a Web page, and an input control.

**Create a Web project**

To create a Web project:

1. Select **File > New > Project > Dynamic Web Project** from the menu. (See Figure 1.)
2. In the New Project wizard:
   1. Enter a project name.
   2. Select **Faces Project** configuration.
   3. Select **Add project to an EAR**.
   3. Click **Finish**.

**Figure 1. New Dynamic Web Project screen**

![New Dynamic Web Project screen](image)

Create a Web page

To create a Web page:

- Right-click the **project name** in the **Project Explorer**.
- Select **New > Web Page** from the list.
- In the **New Web Page** wizard, enter a page name. (See **Figure 2**.)
- Click **Finish**.
Create an input control
Drag an Input component from the Enhanced Faces Components drawer of the palette onto the page. (See Figure 3.)

Quick and easy steps to add type-ahead
Now that you have an input field, you are ready to enable the type-ahead feature for it by following these few steps (see Figure 4):
1. Click the **Input** field to select it.
2. The **Properties** view updates to show properties of the `inputText` tag. Switch to the **Behavior** page for the `inputText` tag.
3. Select **Enable typeahead option**.

That's all you need to do!

**Figure 4. Enabling the type-ahead feature**

Notice that the display of the Input component changed to indicate that it now supports the type-ahead function, and the Properties view switched to show the type-ahead properties. (See **Figure 5**.)

**Figure 5. Type-ahead feature enabled**

Now that you have an input field with type-ahead support enabled, all that is left to handle is the server-side code that generates the list of suggestions.

**Ways to generate suggestions**

Processing the user's input and generating a list of suggestions is the purpose of the type-ahead functionality. This is really the only part that you need to implement every time you want to add a type-ahead feature.

For the type-ahead JSF component to work, its `value` attribute must be bound to a Map object. When the client requests suggestions from the server, it will call the Map's get method and pass the current content of the input field as a parameter. The get method can then produce a list of suggestions and return it to the client in an ArrayList object.
In JSF terms, this means that you need to create a managed bean that implements a java.util.Map interface and add the code to generate suggestions into the get method. You can put this managed bean into session or even application scope, so that it is created only once.

**Generating suggestions in a managed bean**

To create a new bean:

- Right-click the **Java Resources** folder of the project in the **Project Explorer**.
- Select **New > Package** from the list.
- Enter a package name (for example, beans), and click **Finish**.
- Right-click the **package name** in the **Project Explorer**.
- Select **New > Class** from the context menu.
- In the **New Class** wizard (see Figure 6):
  1. Enter a class name (for example, Suggestions).
  2. Select **java.util.AbstractMap** as the superclass.
  3. Select **Create inherited abstract methods**.
- Click **Finish**.

**Figure 6. New class**

To add this bean as a managed bean:
1. Switch back to the Web page (page1.jsp).
2. In the **Page Data View**, right-click the **Faces Managed Beans** category, and select **New > Faces Managed Bean**. (See **Figure 7**.)
3. In the **Add Bean** dialog:
   1. Enter the bean name.
   2. Enter the bean class.
   3. Select the **session scope**.
4. Click **Finish**.

**Figure 7. New managed bean**

Now that you have created the bean, you need to implement its get method. To keep things simple for this example, you will create a list of 10 suggestions by adding a number to the text that the user typed into the field. The entire bean will look like this:

**Listing 1. Suggestions bean**

```java
package beans;
import java.util.AbstractMap;
import java.util.ArrayList;
import java.util.Set;
public class Suggestions extends AbstractMap {
    public Object get(Object key) {
        ArrayList suggestions = new ArrayList(10);
        String currentValue = key.toString();
        for(int i = 0; i < 10; i++) {
            suggestions.add(currentValue + " " + i);
        }
        return suggestions;
    }
}
```

```java
Now that you have created the bean, you need to implement its get method. To keep things simple for this example, you will create a list of 10 suggestions by adding a number to the text that the user typed into the field. The entire bean will look like this:

**Listing 1. Suggestions bean**

```java
package beans;
import java.util.AbstractMap;
import java.util.ArrayList;
import java.util.Set;
public class Suggestions extends AbstractMap {
    public Object get(Object key) {
        ArrayList suggestions = new ArrayList(10);
        String currentValue = key.toString();
        for(int i = 0; i < 10; i++) {
            suggestions.add(currentValue + " " + i);
        }
        return suggestions;
    }
}
```
```
Your bean and the input component are now ready. You just need to bind them together:

- Switch back to the Web page (page1.jsp).
- Select the **type-ahead component**.
- Switch to the **Properties** view. (See Figure 8.)
- Enter the name of the managed bean that you created into the **Value** field: #{suggestions}

**Figure 8. The type-ahead value set**

Now you can save all of the files and run page1.jsp on the server. When the browser window opens, you will see a single input field. (See Figure 9.) As soon as the user starts typing, a suggestion listbox opens under the input field. It contains 10 options, and each option contains the current value of the input field plus a number from zero to 9.

**Figure 9. Run on server**

The suggestions that you get in this example are not very useful, because we intentionally kept the bean code simple here to demonstrate how a type-ahead component works. In a real application, you would probably want to perform a more sophisticated lookup and match the value with a database, such as a list of employees (if the user was typing a name) or a list of products (if the user was searching in a catalog). You could also hardcode the dataset if it was constant and relatively small, such as the list of states in the United States. Or, you could invoke a Web service to get a list of suggestions from an independent provider.
Generating suggestions from a Web service

A Web service is probably the most common source of suggestions for type-ahead fields. When you use a Web service, you don't need to know or worry about where the data is coming from nor how the logic is implemented. The implementation of the get method in your managed bean can be as simple as calling the Web service with the parameter you received and returning the result as an ArrayList.

There are different kinds of Web services that you can use. You can create your own Web service (this goes beyond the scope of this article), you can use a Web service that is available from another team in your organization, or you can use one that is available to the public. There are quite a few public Web services on the Web. For this exercise, you will use a dictionary lookup service on Aonaware. This service can return a list of English words that start with a certain prefix, which makes it perfect for type-ahead functionality if you want to help users enter English words correctly.

To make your application use this Web service, you need to add the service to the Rational Application Developer IDE, so that it generates all of the necessary Web service classes. Then you change the get method in your Suggestions class to use this service.

Adding a Web service to generate suggestions

To add a Web service to your project, follow these steps:

1. Switch back to the Web page (page1.jsp).
2. In the Page Data View, right-click the Services category and select New > Web Service. (See Figure 10.)
3. In the Web Services Discovery dialog:
   1. Select Web Services from a known URL.
   2. Enter the URL for the Web Service: http://services.aonaware.com/DictService/DictService.asmx?wsdl, and then click Go.
   3. When the Web service appears in the Information box, click Add to Project.
4. In the Add Web Service dialog, select any method, and then click Finish.
Improving usability of Web applications with type-ahead input fields using JSF, AJAX, and Web services in Rational Application Developer V7

Figure 10. Add a Web service

Using the Web service

After you have added the Web service to your project, the Rational Application Developer IDE generates all of the code that is required to access the service from your bean. You can now go back to the Suggestions class and modify the get method to invoke the service, so you don't get 10 useless strings.

The code for the Suggestions bean looks like this:

Listing 2. Suggestions bean with a Web service

```java
01 package beans;
02
03 import java.util.AbstractMap;
04 import java.util.ArrayList;
05 import java.util.Set;
06 import com.aonaware.services.DictServiceSoapProxy;
07 import com.aonaware.services.DictionaryWord;
08
09 public class Suggestions extends AbstractMap {
10
11 DictServiceSoapProxy proxy = null;
12
13 public Suggestions() {
14  proxy = new DictServiceSoapProxy();
15 }
16
17 public Object get(Object key) {
18  String currentValue = key.toString();
19  if(currentValue.length() < 3) {
20   return null;
21  } else {
22    ArrayList result = new ArrayList();
```
try {
    DictionaryWord[] words = proxy.matchInDict("wn", currentValue, "prefix");
    for(int i = 0; i < words.length; i++) {
        String word = words[i].getWord();
        result.add(word);
    }
} catch (Exception e) {
}
return result;

public Set entrySet() {
    return null;
}

Here's a brief analysis of what key parts of this code do:

- In line 11, you declare a proxy variable to use to invoke the service. (The DictServiceSoapProxy class is one of the classes that Rational Application Developer software generated when you added the service to the project).
- In line 14, you create an instance of the Web service proxy.
- In lines 19-21, you make sure that the string you are about to pass to the service is not too short. This is to save the server from extremely large requests. (Imagine inadvertently asking it to return all words that start with an A!)
- In line 24, you invoke the service and pass the current value of your input field as a parameter. The other two parameters (wn and prefix) define which dictionary you want to use and what matching rule you want to apply.
- In lines 25-28, you convert the result returned by the service into an ArrayList that the type-ahead component understands.

Now you are ready to save all of the files and run page1.jsp on the server again. (See Figure 11.) This time, nothing will happen, as long as you typed only one or two letters. But as soon as you type at least three letters, the suggestion box will open with a list of the first 10 words returned from the dictionary.

**Figure 11. Run on server**

<table>
<thead>
<tr>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td>test</td>
</tr>
<tr>
<td>test ban</td>
</tr>
<tr>
<td>test bad</td>
</tr>
<tr>
<td>test case</td>
</tr>
<tr>
<td>test copy</td>
</tr>
<tr>
<td>test drive</td>
</tr>
<tr>
<td>test driver</td>
</tr>
<tr>
<td>test equipment</td>
</tr>
<tr>
<td>test fly</td>
</tr>
<tr>
<td>test instrument vehicle</td>
</tr>
</tbody>
</table>

**How to customize your type-ahead component**

You can customize the behavior of the type-ahead by using these attributes:
### Attribute Table

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxSuggestions</td>
<td>The maximum number of suggestions displayed. Default is 10.</td>
</tr>
<tr>
<td>startCharacters</td>
<td>The minimum number of characters that must be in the input field before a suggestion is made. Default is 1.</td>
</tr>
<tr>
<td>startDelay</td>
<td>Number of milliseconds delay before a suggestion is made. Default is 500.</td>
</tr>
<tr>
<td>matchWidth</td>
<td>If true, the suggestion list is as wide as the input field. If false, its width is determined by CSS of the list.</td>
</tr>
<tr>
<td>size</td>
<td>The height (in lines) of the suggestion list. If there are more suggestions than lines, scrolling is turned on in the suggestion list. Default is 10.</td>
</tr>
<tr>
<td>type</td>
<td>suggest or autocomplete specify whether suggestions are displayed in a drop-down list (the user chooses from the list) or, instead, the field auto-completes with the first suggestion in the list (in this case, the user can use arrow keys to choose a different suggestion). Default is suggest.</td>
</tr>
<tr>
<td>onstart</td>
<td>JavaScript function that is executed when the AJAX request is submitted.</td>
</tr>
<tr>
<td>oncomplete</td>
<td>JavaScript function that is executed when the AJAX request is processed.</td>
</tr>
</tbody>
</table>

### Note:
You can use onstart and oncomplete to give the user visual feedback that the service is fetching suggestions, for example.

The presentation of the suggestion list box is controlled by these CSS classes:

<table>
<thead>
<tr>
<th>CSS class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;base&gt;</td>
<td>The name of this class can be set in the styleClass attribute of the type-ahead component. Default is inputText_Typeahead. Styles the div that contains the list.</td>
</tr>
<tr>
<td>&lt;base&gt;-List</td>
<td>Styles the window that displays the suggestion list.</td>
</tr>
<tr>
<td>&lt;base&gt;-Item</td>
<td>Styles individual items in the list.</td>
</tr>
<tr>
<td>&lt;base&gt;-Item-Selected</td>
<td>Styles the currently selected item.</td>
</tr>
</tbody>
</table>

### A word of caution

The type-ahead component, like any other component that uses AJAX, has the potential to generate a lot of network traffic between the browser and the server. When considering this component for your Web application, keep in mind that every user could be generating additional requests with every keystroke. If you are serving a lot of users, this could be a lot of requests for the server to handle.

To limit the number of requests, it is best to set the startDelay and startCharacters attributes to relatively high values, so that the browser sends AJAX requests only after a significant delay -- either after the user types a few characters or after the user pauses for a while.
Related topics

- Build your next development project with IBM trial software, available for download directly from developerWorks.
- Visit the Rational Application Developer product page to expand your product skills.
- Download a free trial version of Rational Application Developer

© Copyright IBM Corporation 2006
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)