RapidSpell Web

v3 Java Edition

Software Component

User's Guide

Keyoti

Contents

Terms & Conditions

If you do not agree to these terms and conditions then you may not install, use, or distribute RapidSpell Web.

1. License and Ownership RapidSpell Web is a licensed software product. RapidSpell Web, including its accompanying files and documentation, is owned and copyrighted by Keyoti Inc., © Copyright 2003-2008, all rights reserved. Keyoti Inc. grants the user in possession of an official receipt of purchase a nonexclusive license to download and use the software, for any lawful commercial or non-commercial purposes provided the terms of this agreement are met.

2. Distribution. If you purchased an Individual License Version of RapidSpell Web, you many copy and use the RapidSpell Web distribution file (.zip) on any systems you use but for your individual use only. Individual use is defined as one person having the ability to read or copy any of the files originally contained in the RapidSpell distribution file including (but not limited to) the .jar, .java, .html, .gif, .jpg, .bat, and .txt files. You must take appropriate safeguards to protect this product from unauthorized access by others.
If you purchased a Site License Version of RapidSpell Web, then you may provide copies of the RapidSpell Web distribution file (.zip) to anyone employed by your company whose primary work address is within a 100 mile (160 km) radius of your primary work address. If you purchased an Enterprise License Version then you have the same rights as the Site License with, however, no geographical or numerical restrictions on the locations to which you may provide copies of the RapidSpell Web distribution file. These individuals then have license rights and responsibilities equivalent to the Developer License Version.
In all cases the copies of the RapidSpell Web distribution file (.zip) must be unaltered and complete, including this license agreement and all copyright notices. Evaluation version must not be re-distributed in any form.
You may not reverse engineer, disassemble, decompile, or modify this software or its accompanying documentation in any way.
You may reproduce and redistribute a copy of the jar files as part of any Java based software developed and licensed by you that itself uses RapidSpell Web provided
a) your software is installed on no more web-site domains or servers than specified by the license type purchased,
b) your software has clearly distinct and added functionality,
c) you are responsible for all technical support,
d) the RapidSpell Web application programming interfaces are not documented, exposed, or otherwise made available by your software,
e) attribution is given to Keyoti Inc. (http://www.keyoti.com) in any documentation or screen displays where other credits appear,
f) you do not allow recipients of your software to reverse engineer, disassemble, decompile, or copy portions from your software allowing them to gain separate access to RapidSpell Web or any parts of it.
The source code samples included with this package and in the RapidSpell Web documentation may be freely used, modified, incorporated, and distributed without restriction as part of any software that uses RapidSpell Web itself,
g) your software is not used by software or web developers as part of their application, and
h) your software is not a software component or JSP Tag.

3. Warranty Disclaimer and Limitation of Liability. RapidSpell Web is licensed to the user on an "AS IS" basis. Keyoti Inc. MAKES NO WARRANTIES, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THIS SOFTWARE AND ITS ASSOCIATED FILES AND DOCUMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. KEYOTI INC. DOES NOT WARRANT THAT THE OPERATION OF THIS SOFTWARE WILL BE UNINTERRUPTED OR ERROR FREE, OR THAT DEFECTS WILL BE CORRECTED. You the user are solely responsible for determining the appropriateness of this software for your use and accept full responsibility for all risks associated with its use. Keyoti Inc. is not and will not be liable for any direct, indirect, special or incidental damages in any amount including loss of profits or interruption of business however caused, even if Keyoti Inc. has been advised of the possibility of such damages. Keyoti Inc. retains the right to, in its sole discretion, refund the purchase price of this software as a complete and final resolution to any dispute.

SPECIFIC DISCLAIMER FOR HIGH-RISK ACTIVITIES.The SOFTWARE is not designed or intended for use in high-risk activities including, without restricting the generality of the foregoing, on-line control of aircraft, air traffic, aircraft navigation or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. Keyoti Inc. and its suppliers specifically disclaim any express or implied warranty of fitness for such purposes or any other purposes.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. To the maximum extent permitted by applicable laws, in no event shall Keyoti Inc. or its suppliers be liable for any damages whatsoever (including without limitation, damages for loss of business profits, business interruption, loss of information, or other pecuniary loss) arising out of the use of or inability to use this Keyoti Inc. product, even if Keyoti Inc. has been advised of the possibility of such damages. Because some states and jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

4. Support and Source Code Licensing. Keyoti Inc. does provide formal customer support contracts for RapidSpell Web. We also make every effort to ensure the quality of our products, and questions, bug reports, and feature requests are encouraged, please use email. Due to resource limitations, it may not be possible to resolve all issues and no assurances can be given as to when or if problems will be addressed. Customers incorporating RapidSpell Web into software for deployment or sale are encouraged to purchase the Source Code License Version of RapidSpell Web which will enable self-support, debugging, and modifications or extensions to the base functionality (but not distribution of the source code itself).

Introduction

Thank you for purchasing RapidSpell Web, please keep up to date through our website keyoti.com and feel invited to post feedback. We also ask that you register your purchase so that we may offer you support and free upgrades from time to time, at http://keyoti.com/care/register

Online Resources

We invite you to visit our Knowledge Base and Support forum at http://keyoti.com/kb and http://keyoti.com/support.

Overview

RapidSpell Web provides a spelling component to add spell checker functionality to your web applications by simply adding the Tag to a JSP page or using the Beans in Servlets. There are no programming skills required to use RapidSpell Web Tags, making it accessible to JSP page designers as well as Java programmers. The UI provides all the usual features, add, change, change all, ignore, ignore all, undo, and smart suggestions. It interactively checks any HTML text box component such as <textarea> and <input type='text'> aswell as 3rd party rich Html text box components. The spell checker UI features an advanced preview pane to interactively highlight errors (with no post back). The spell checker accepts manual corrections, and also supports user dictionaries (either one dictionary for all users or separate user dictionaries). The included non GUI component provides core spell checker functionality at the middleware level, which means it is suitable for server applications as well as any console application.

Common Uses

The Tag components can check any Html text component (e.g., <textarea>, <input type='text'> and 3rd Party).

The non-GUI component provides core spell checking functionality, allowing it to be useful in any application, including: Server applications and Console based applications

Requirements

The components were built to JSP1.2 specifications, they were developed and tested on Tomcat 4+, IBM WebSphere and Bea WebLogic.

Release Notes

Please see the release notes text file in the ZIP for a list of improvements and compatibility info.

The Dictionaries

Included are 4 dictionary assemblies, UK and US English dictionaries, each containing ~150K words, a combined UK and US dictionary containing ~155K words. These can be found in the 'dictionaries' directory.

Installation & Deployment

1. To use RapidSpell Web, you must add the RapidSpellWeb.jar and RapidSpellMDict.jar files to the lib directory of your web application.

2. To use the inline spell checker tags (RapidSpellWInline, RapidSpellWInlineTextBox and RapidSpellWInlineHelper), you must register a Servlet in your web.xml file. To do this;

i) Open web.xml under /web-inf

ii) Add this block (if other <servlet> sections exist, add it directly below them)

```
<servlet>
<description>Handles requests for RapidSpell Web</description>
<servlet-name>RapidSpellWebHandlerServlet</servlet-name>
<servlet-class>com.keyoti.rapidSpell.web.ControlServlet</servlet-class>
<init-param>
      <param-name>licenseKey</param-name>
      <param-value></param-value>
</init-param>
</servlet>
```

iii) Add this block below it (if other <servlet-mapping> sections exist, add it directly below them)

```
<servlet-mapping>
      <servlet-name>RapidSpellWebHandlerServlet</servlet-name>
      <url-pattern>*.rapidspellweb</url-pattern>
</servlet-mapping>
```

iv) For an example of a complete web.xml file with this, please see the demo application included with the product.

3. Add the taglib directive to the top of each page using the tags.

<%@ taglib uri="http://www.keyoti.com/" prefix="RapidSpellWeb" %>

Licensing The Components

RapidSpell Web requires a license key to be set in the application, **if a key is not set, the component will only run when the application is accessed through "localhost".** Furthermore, if the RapidSpellChecker class is used separately, it too must have a license key set (in it's constructor).

Evaluation Keys

Time limited evaluation keys can be generated automatically at
http://keyoti.com/products/evaluation-key-generator

Purchased Keys

If you have purchased licenses, please see the accompanying file "license-keys.txt" which explains in detail how to generate "deployment keys".

Setting A License Key

The license key (deployment key) is set as follows, depending on which component is being used.

RapidSpellWeb (Dialog Spell Checker)

The key can be set in the licenseKey attribute in the RapidSpellWeb Tag (which is in the popup page), eg;

```
<RapidSpellWeb:rapidSpellWeb licenseKey="......."/>
```

RapidSpellChecker (Programmatic Spell Checker)

The key can be set in RapidSpellChecker's constructor, eg;

```
RapidSpellChecker rapidSpellChecker = new RapidSpellChecker(".......");
```

RapidSpellWInline (Inline Spell Checker)

This component can be used/licensed in 2 ways and the key is set accordingly:

1. The standard/default way (when the rapidSpellWInlineHelperPage attribute in each RapidSpellWInline tag is not set in the JSP page) -

Open web.xml under /web-inf and place the license key inside the <param-value> tags, for the licenseKey property. Eg.

```
<init-param>
<param-name>licenseKey</param-name>
<param-value>12345</param-value>
</init-param>
```

(the web application server, or 'container' must be restarted after this change).

2. If rapidSpellWInlineHelperPage is set in the RapidSpellWInline tag -

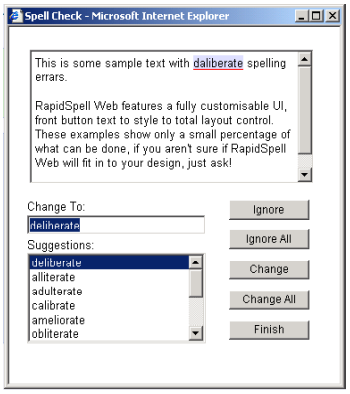
It is necessary to set the licenseKey attribute of the RapidSpellWInlineHelper tag (which is in the page specified by the rapidSpellWInlineHelperPage property).

eg;

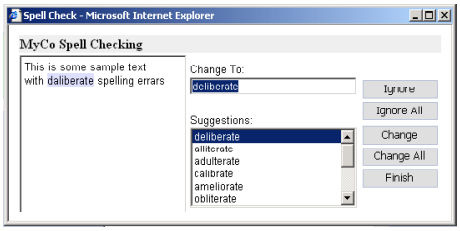
```
<RapidSpellWeb:rapidSpellWInlineHelper licenseKey="......."/>
```

The Components

RapidSpellWeb (Dialog Spell Checker)



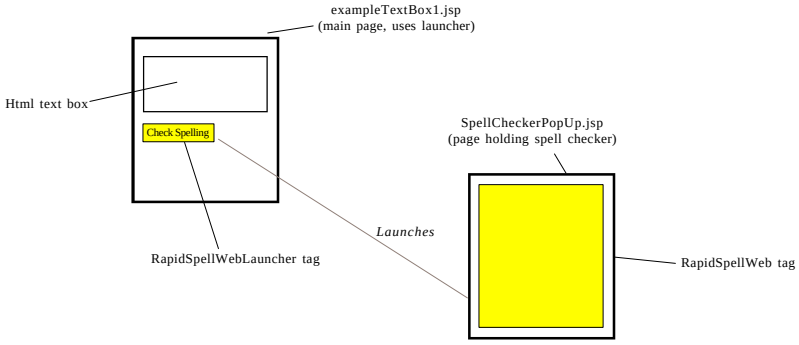
Standard Layout



Custom Layout

The RapidSpell Web Spell Checker is actually used via two Tags (and optionally a third for multiple text box checking), called RapidSpellWebLauncher and RapidSpellWeb, which launch the spell check page and show the spell checker, respectively.

RapidSpellWebLauncher creates a form button, which will either open a popup window or submit the form (depending on the 'mode' used). In the new page that is loaded the RapidSpellWeb is embedded to display the spell checker.



The RapidSpellWeb Tag runs through the entirety of the HTML text box specified in the RapidSpellWebLauncher tag parameters. The user is shown the spell checker page, which will iterate through all the misspelt words in the text, highlighting them in the 'preview pane' and allowing the user to ignore, change, add etc. When the text has been checked the user will be prompted that the spell check has finished, and upon clicking OK, the page will return to the one specified in the CallBack parameter of RapidSpellWebLauncher.

RapidSpellWeb Summary

The RapidSpellWeb Tag is the actual spell checker, it's properties may be set to define the layout, style and text of the component. Here is a summary of the main features of the RapidSpellWeb Tag.

Incorrectly spelt words are highlighted in the preview pane as the user is queried.
Server side user dictionaries
Fully customizable user-dictionary at development and install time.
Fully customizable UI layout, style and text.

Key Control Properties

For explicit details of all available properties please consult the API documentation and later sections of this guide.

layout - Optional HTML template used to layout the control, identifying tags are used to denote positions of buttons, labels and fields.
ignoreButton, ignoreAllButton, changeButton, changeAllButton, undoButton, finishButton, addButton (if user dictionary is specified) - All have definable style and text.
changeToLabel, suggestionsLabel - Labels with definable style and text.
suggestionsBox - Definable style and text.
changeToBox - Definable style and text.
badWordHighlightStyle - A CSS style string to define the style of the bad word highlighting, eg. "border-bottom: 1px solid red; background-color: #ddddff"
previewPaneStyle - A CSS style string allowing definition of the preview pane style

RapidSpellWebLauncher Summary

This component should be integrated in to the form containing the text area to be checked, it is rendered visibly as a button but contains hidden form fields and Javascript to open the page containing the RapidSpellWeb control, this Tag's properties are used to define the behaviour of the spell checker, it features

Customizable style and text for the button.
Two modes of operation of the spell checker, popup or separate.
Defines the behaviour of the spell checker.

Key Control Properties

For more details please consult the API documentation and later sections of this guide.

suggestionsMethod - the suggestions method used by RapidSpell, either

"HASHING_SUGGESTIONS" (default) or "PHONETIC_SUGGESTIONS"
includeUserDictionaryInSuggestions - "true" or "false", if the user dictionary is defined this will set whether the suggestions should come from the user dictionary as well, has a performance overhead proportional to the size of the user dictionary, false by default.
ignoreCapitalizedWords -"true" or "false", whether to ignore words that start with capital letters, false by default. This improves performance marginally if true.
considerationRange - An integer greater than 1 that defines how large a word range to consider for suggestions, has considerable affect on performance, consult advanced topics in this manual for more info.
mode - "popup" for the spell checker to be opened in a small new window, "separate" to open the spell checker in the main window on a new page.
userDictionaryFile - The full path of the file on the SERVER to be used as the user dictionary, does not have to exist.
windowWidth - Int width of the popup window, only relevant in popup mode
windowHeight - Int height of the popup window, only relevant in popup mode
rapidSpellWebPage - The URL of the page holding the RapidSpellWeb control
callBack - In 'separate' mode this is the URL of the page the corrected text should be posted to after spell checking, in 'popup' mode this is the form and element name of the text area to put the text in.
textComponentName - The form and name of the element holding the text to be checked.
buttonStyle/Text - The button style and text.
ignoreXML - true/false, whether to ignore XML/HTML tags (false by default)
textComponentInterface - The type of text component to interface with, 'Standard', 'RichTextBox' or 'Custom'
finishedListener - The name of a Javascript function in the main form that should be called when the pop up checker finishes
separateHyphenWords - true/false, whether to treat hyphenated words as separate words (false by default)

RapidSpellWebMultiple Summary

This component works with RapidSpellWebLauncher Tags to enable simple multiple text box checking. By working with RapidSpellWebLauncher or RapidSpellWInline, each text box can have it's own spell check options set. This Tag's members are, for the most part, the same as RapidSpellWebLauncher since both controls are used in the same manner (to launch spell checking). Please see the multiple text box example JSP in the demos directory and the example usage in this guide, for more information.

RapidSpellChecker (Programmatic Spell Checker)

Provides a model (business/logic) level spell checker API which can be used in a client-server environment and any applications where a client GUI does not exist. RapidSpellChecker has been benchmarked at checking 50,000 words/second on a standard 1GHz PC. User dictionaries can optionally be used, allowing words to be added. This component can be used in 2 different ways, in an iterative fashion or on a query by query basis. Please see the API docs for code level details.

Using RapidSpellChecker in an iterative manner provides complete textual correction of strings similar to a GUI based spell checker. A string is loaded into RapidSpellChecker and then successive calls to the nextBadWord() method move through the string allowing findSuggestions() and changeBadWord(replacement) to be called to find spelling suggestions and optionally correct the misspelt words in the text, finally getAmendedText() returns the corrected text.

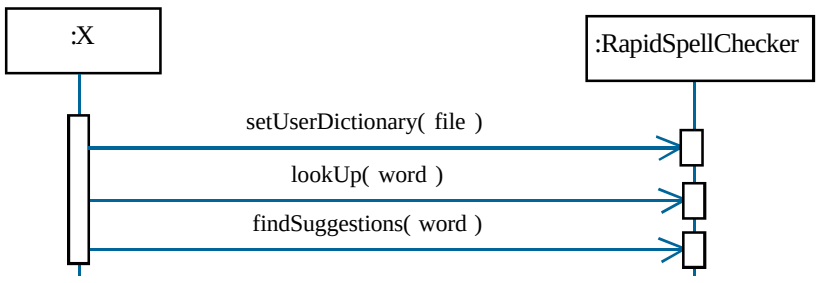
Alternatively RapidSpellChecker may be used in a simple query by query basis, calls to lookUp(word) and findSuggestions(word) methods allow bare bones spell checking functionality.

Key API Points

Methods

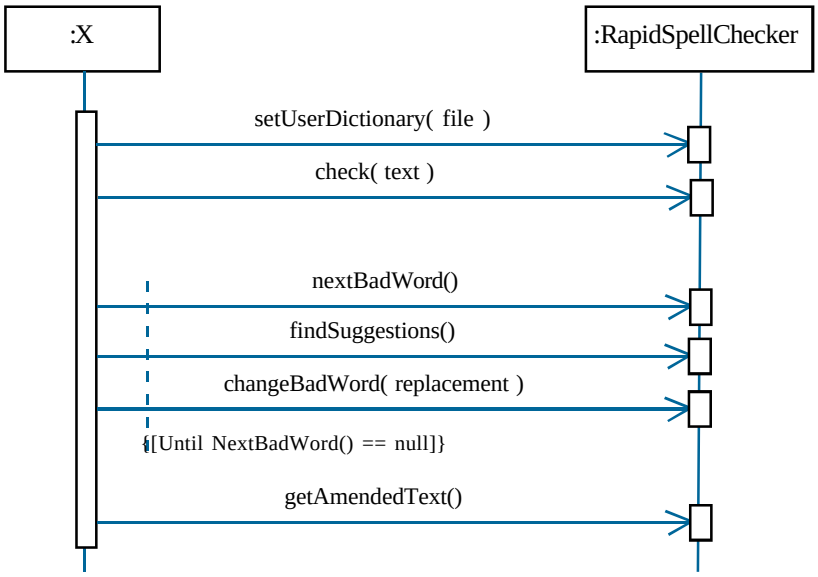
RapidSpellChecker()	-argumentless constructor
check(text [, startPosition])	-begins checking text
findSuggestions(word)	-finds suggestions for word
lookUp(word)	-checks if word is in either main or user dictionaries
setUserDictionary(file)	-allows you to define which file will be used as a user dictionary
ignoreAll(word)	-ignores all future occurrences of word
addWord(word)	-adds word to the user dictionary if it exists
nextBadWord()	-finds the next misspelt word in the text
changeBadWord(newWord)	-changes the misspelt word last identified by NextBadWord()
getAmendedText()	-returns the corrected text

Query by Query Usage Sequence Diagram



Iterative Usage Sequence Diagram

See examples section for examples.



RapidSpellWInline (Inline/As-You-Type Spell Checker)

This style of spell checking is close to Windows 'as-you-type' spell checking. Of course true as-you-type checking is near impossible on the web without plug-ins or other client side technologies, however this control does provide user friendly in-document highlighting, without plug-ins or other special settings. The core component to this type of checking is the RapidSpellWInline control, which is similar in purpose to the RapidSpellWebLauncher control. It shows as a button on the page, and is configured to check a text box through it's TextComponentID property.

Besides this core component there is also the RapidSpellWInlineTextBox control, which is an optional replacement to the standard text box. RapidSpellWInlineTextBox functions almost identically to a standard text box, except it also allows the spell checker to highlight errors in it while still in edit mode. If the RapidSpellWInline control is configured to check a standard text box then it will cause the text box to become 'static' while the errors are highlighted. It is compatible with Internet Explorer 5+, Netscape 6+, Firefox 1, Opera, and Safari on the PC and Mac.

RapidSpellWInline

Sits on the same page as the text box and renders as a button or image button which the user can use to trigger spell checking. Can also function with RapidSpellWebMultiple, and can be made invisible for Javascript access.

Highlights errors in the text box, provides a context menu with suggestions, Ignore All, Change All (optional) and Add (optional).

RapidSpellWInlineHelper

Optional: This control can be placed on an empty page to allow the behavior of the spell checker to be changed. By default this is embedded in the RapidSpellWeb jar and the rapidSpellWInlineHelperPage property does not have to be set.

Client Files

By default, client files are served from inside the Jar via the RapidSpellWebHandlerServlet. To use external client files (that you can modify) specify the following static member at the top of the user JSP (or in a Servlet), ensure that the path you use has a trailing slash.

```
<%
com.keyoti.rapidSpell.web.Globals.clientFilesLocation = "/myApp/resources/Keyoti/
ClientFiles/";
%>
```

and copy the client files to the folder specified above (changing the path as required).

RapidSpellWInlineTextBox

Optional: Replacement for a standard text box, when RapidSpellWInline is used with this control (and the browser is IE5.5+ or Mozilla 1.4+ - FF, NS7.1) the errors are shown inside the textbox while it is still editable (when the browser is older, the control behaves exactly as a standard text box). This control is necessary for as-you-type spell checking.

RapidSpellWebMultiple

This component works with RapidSpellWebLauncher Tags to enable simple multiple text box checking. By working with RapidSpellWebLauncher or RapidSpellWInline, each text box can have it's own spell check options set. This Tag's members are, for the most part, the same as RapidSpellWInline since both controls are used in the same manner (to launch spell checking). Please see the multiple text box example JSP in the demos directory and the example usage in this guide, for more information.

Dict Files

RapidSpell now supports it's own proprietary dictionary format, these dictionaries are called Dict files. They have been developed to allow more flexibility than the existing dictionary format. The existing MDict jar format will continue to be supported, the advantages and disadvantages of the formats are outlined below:

New Dict File;

Advantages
- Swappable during run-time (necessary for other languages)
- Customizable by developer (future)

Disadvantages
- Larger file size

Existing MDict.jar format;

Advantages
- Compressed

Disadvantages
- Slower

If it is necessary to minimise the dictionary size then the Jar format should be used, however for all other uses the new Dict File format is preferable.

Using Dict Files

The RapidSpellMDict.Jar file must reside in the same directory as the main RapidSpell Jar, it will be used if the dictFile property is null. To use a Dict file simply set the dictFile attribute (of RapidSpellWebLauncher or RapidSpellWInline) to the absolute file path of the Dict file to be used.

Dictionary Customization

It is possible to customize the dictionaries and create new ones using the convenient, free Dict Manager tool included in the full version of RapidSpell Web.

Examples (Dialog/PopUp Spell Checker)

Below are some use scenarios of the 2 spell checker components supplied, please refer to the API documentation for additional detail.

Simple Popup Spell Checking

To use RapidSpell Web to spell check a web page in popup mode, create a page holding the RapidSpellWeb Tag, then add the RapidSpellWebLauncher Tag to your existing page with the text element you want checked, referencing the page you created with RapidSpellWeb in the rapidSpellWebPage property.

Example, note this code is highly simplistic for demonstration purposes.

Code Example RapidSpellCheckerPopUp.jsp

Page containing the RapidSpellWeb Tag

```
<%@ taglib uri="http://www.keyoti.com/" prefix="RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<title>Spell Check</title>
<body>
<Center>
<RapidSpellWeb:rapidSpellWeb/>
</center>
</body>
</html>
```

Code Example RSWL-Simple.jsp

Page containing the RapidSpellWebLauncher Tag, where the text element to be checked is located.

```
<%@ taglib uri="http://www.keyoti.com/" prefix="RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<BODY >
 <%
 if(request.getParameter("fMessage") == null){
 %>
       <form action='RSWL-Simple.jsp' method='post' name='myForm' id='myForm'>
            <input type='hidden' name='fMessage' value='complete'>
            <textarea name="sourceTextBox" wrap='true' cols='40' rows='10'>This
is some sample text with daliberate spelling errars</textarea>
            <br>
```

```
            <!-- Creates a 'Check Spelling' button,
                  TextComponentName = name of textarea to check,
                   -->
            <RapidSpellWeb:rapidSpellWebLauncher
                              rapidSpellWebPage="RapidSpellCheckerPopUp.jsp"
                              textComponentName="myForm.sourceTextBox"
                              separateHyphenWords="false"
                              userDictionaryFile="/dictionary.txt"
                              />

            <input type='submit'>
       </form>
 <%
 } else {
 %>
       Text entered was:
       "<%=request.getParameter("sourceTextBox")%>"
 <%
 }
 %>
</body>
</HTML>
```

The main page (RSWL-Simple.jsp) uses the RapidSpellWebLauncher Tag to create the check spelling button, it's properties specify which form element to check (textComponentName), the URL of the page containing the RapidSpellWeb control (rapidSpellWebPage) and the mode (mode). Note that (for mode='popup') the properties textComponentName and callBack refer to form elements in Javascript notation, relative to the document, i.e. they should always have the name of the form (or an element from the forms array) followed by the text field name. RapidSpellWebPage can be a relative or absolute URL.

Setting Style

It is very simple to define the style properties of the HTML elements in the Tags, eg;

Code Snippet Button Customization

```
ignoreButtonStyle="font-family:'Tahoma,Arial,Helvetica';font-size:10pt;
border:1px solid #b5bed6; background-color:#dddddd; width: 90px;"
ignoreButtonOnMouseOver="this.style.backgroundColor='#b5bed6';this.style.borderColor='#08246b';"
ignoreButtonOnMouseOut="this.style.backgroundColor='#dddddd';this.style.borderColor='#b5bed6';"
```

This specifies the style using regular CSS syntax and also specifies onMouseOver/Out Javascript

events to modify the style for an attractive effect.

Separate Page Spell Checking

To use RapidSpell Web to spell check a web page in separate mode, create a page holding the RapidSpellWeb Tag, then add the RapidSpellWebLauncher Tag to your existing page with the text element you want checked, referencing the page you created with RapidSpellWeb in the RapidSpellWebPage property.

Code Example From SpellCheckSeparatePage.jsp

Page containing the RapidSpellWeb Tag.

```
<%@ taglib uri="http://www.keyoti.com/" prefix="RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
   <HEAD>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
      <style>.....</style>
      </HEAD>
   <BODY ...>
<!-- Any HTML formatting can go here -->
<RapidSpellWeb:rapidSpellWeb
ignoreButtonStyle="...."
ignoreButtonOnMouseOver="...."
ignoreButtonOnMouseOut="...."
ignoreAllButtonStyle="...."
ignoreAllButtonOnMouseOver="...."
ignoreAllButtonOnMouseOut="...."
....
..Other button style definitions..
....
/>
</BODY>
</HTML>
```

Code Example From RSWL-Separate.jsp

Page containing the RapidSpellWebLauncher Tag

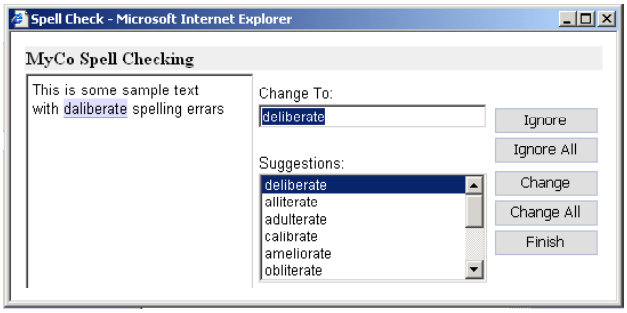
```
<%@ taglib uri="http://www.keyoti.com/" prefix="RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
   <HEAD>
      <style>...</style>
      </HEAD>
   <BODY ...>
         <%
```

```
          if(request.getParameter("fMessage") == null){
         %>

      <p>This demo page opens the spell checker in a separate page.
      <form action='RSWL-Separate.jsp' method='post' name='myForm'>
           <input type='hidden' name='fMessage' value='complete'>

             <% if(request.getParameter("RSCallBack") == null){    %>
                   <textarea name="sourceTextBox" wrap='true' cols='40'
rows='10'>This is some sample text with daliberate spelling errars</textarea>
             <% } else { %>
                   <textarea name="sourceTextBox" wrap='true' cols='55'
rows='10'><%= request.getParameter("CorrectedText") %></textarea>
             <% } %>
                                <br>

             <!-- Creates a 'Check Spelling' button, TextComponentName = name of
textarea to check, CallBack = name of to put corrected text -->

             <RapidSpellWeb:rapidSpellWebLauncher
                  rapidSpellWebPage="SpellCheckSeparatePage.jsp"
                  textComponentName="myForm.sourceTextBox"
                  callBack="RSWL-Separate.jsp"
                  separateHyphenWords="false"
                  mode="separate"
                  buttonStyle="font-family:'Tahoma,Arial'; border:1px solid
#b5bed6; background-color:#dddddd;"

buttonOnMouseOut="this.style.backgroundColor='#dddddd';this.style.borderColor='#b5bed6';"

buttonOnMouseOver="this.style.backgroundColor='#b5bed6';this.style.borderColor='#08246b';"
             />

             <input type='submit' Style="font-family:'Tahoma,Arial'; border:1px
solid #b5bed6; background-color:#dddddd;"
onMouseOut="this.style.backgroundColor='#dddddd';this.style.borderColor='#b5bed6';"
onMouseOver="this.style.backgroundColor='#b5bed6';this.style.borderColor='#08246b';"
>
    </form>


                <%
        } else {
        %>

             Text entered was:
             "<%=request.getParameter("sourceTextBox")%>"
```

```
        <%
        }
          %>
</BODY>
</HTML>
```

In these pages the style of the buttons has been set through the properties of the controls. Note that the callBack is set to the URL of the page the user should be directed to after they have finished checking the document, this page is called with the parameter CorrectedText which holds the text after the spell check.

Customising RapidSpellWeb Layout



It is possible to customise the layout of the spell checker using a HTML template mechanism. This is done by specifying the layout property with a string of Html using special tags to indicate the position of the essential spell checker elements. The tags are <PreviewPane/>*, <AddButton/>, <IgnoreButton/>, <IgnoreAllButton/>, <ChangeButton/>, <ChangeAllButton/>, <FinishButton/>, <UndoButton/>, <ChangeToLabel/>, <ChangeToBox/>*, <SuggestionsLabel/>, <SuggestionsBox/>* - tags marked with * are required for correct functionality.

Code Example From CustomSpellCheckerPopUp.jsp With Customized Layout

Page containing the RapidSpellWeb Tag

```
<%@ taglib uri="http://www.keyoti.com/" prefix="RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<title>Spell Check</title>
<body>
<Center>

<RapidSpellWeb:rapidSpellWeb
```

```
      changeToBoxStyle="font-size:10pt; width:200px; font-family:'arial,
helvetica, sans-serif';"
      changeToLabelStyle="font-size:9pt; font-family:'arial, helvetica, sans-
serif'; "
      suggestionsBoxStyle="font-size:10pt; width:200px; font-family:'arial,
helvetica, sans-serif';"
      suggestionsLabelStyle="font-size:9pt; font-family:'arial, helvetica, sans-
serif';"
      previewPaneStyle="font-family: 'arial, sans-serif, helvetica'; font-size:
8pt; font-weight:bold;"
      previewPaneWidth="200"
      previewPaneHeight="190"
      ignoreButtonStyle="font-family:'Tahoma,Arial,Helvetica';font-size:10pt;
border:1px solid #b5bed6; background-color:#dddddd; width: 90px;"

ignoreButtonOnMouseOver="this.style.backgroundColor='#b5bed6';this.style.borderColor='#08246b';"

ignoreButtonOnMouseOut="this.style.backgroundColor='#dddddd';this.style.borderColor='#b5bed6';"

      .....button styles.....

finishButtonOnMouseOut="this.style.backgroundColor='#dddddd';this.style.borderColor='#b5bed6';"

      layout="
            <table border=0>
            <tr><td colspan=3 bgcolor='#eeeeee'><b>MyCo Spell Checking</b></td></
tr>
            <tr><td>
                  <PreviewPane/>
            </td>

            <td>
                  <table border=0 height=190>
                        <tr><td>
                              <ChangeToLabel/>
                        </td></tr><tr><td>
                              <ChangeToBox/>
                        </td></tr><tr><td>
                              <SuggestionsLabel/>
                        </td></tr><tr><td valign=bottom>
                              <SuggestionsBox/>
                        </td></tr>
                  </table>
            </td>

            <td>
```

```
                  <table border=0 height=160>
                        <tr><td>
                              <IgnoreButton/>
                        </td></tr><tr><td>
                              <IgnoreAllButton/>
                        </td></tr><tr><td>
                              <AddButton/>
                        </td></tr><tr><td>
                              <ChangeButton/>
                        </td></tr><tr><td>
                              <ChangeAllButton/>
                        </td></tr><tr><td>
                              <FinishButton/>
                        </td></tr>
                  </table>
            </td>
            </tr>
            </table>
            "
/>

</center>
</body>
</html>
```

The only restriction is that the special tags must be on a line by themselves.

Specifying User Dictionaries

It is possible to use user dictionaries with RapidSpell Web, on a one user dictionary for all users, or one user dictionary per user or one user dictionary per group basis. The user dictionary file is specified in the userDictionaryFile property of the RapidSpellWebLauncher or RapidSpellWInline tag. This property specifies the path to the user dictionary file on the server, the developer has total control over which dictionary file is used for a user. Therefore it is up to the developer to manage a user's session and retrieve the path to a particular user's/group's dictionary file, if required.

Using RapidSpell Web With 3rd Party Components

3rd Party text components can be used with RapidSpell Web. Html text box support is built in, but other types of text components can be used if they allow Javascript to access their text. To use with an Html text box simply set ignoreXML to true (this will mean HTML tags are ignored), set TextComponentInterface to "HTMLTextBox" and set TextComponentName to the name of the component.

Code Example Extract From RSWL-3rdParty.jsp Using An Html Text Box

Page containing the RapidSpellWebLauncher Tag

```
.....................
<RapidSpellWeb:rapidSpellWebLauncher
      rapidSpellWebPage="RapidSpellCheckerPopUp.jsp"
      textComponentName="HTB"
      ignoreXML="true"
      textComponentInterface="HTMLTextBox"
/>
.....................
```

To use RapidSpell Web with other 3rd party components it is necessary to write a custom Javascript interface which will perform two tasks for RapidSpell Web, it must get the text from the control and set text to the control. To enable the use of a custom interface the TextComponentInterface property should be set to "Custom", and a Javascript (ECMAscript) block should be written into the form containing RapidSpell Web (Launcher). As an example, a custom interface for the standard HTML text area will be built, to demonstrate the concepts.

Firstly the RapidSpellWebLauncher control's TextComponentInterface is set to Custom:

```
<RapidSpellWeb:rapidSpellWebLauncher id="rapidSpellWebLauncher"
            textComponentName="myForm.tA"
            mode="popup"
            textComponentInterface="Custom"
            ignoreXML="true"
            rapidSpellWebPage="RapidSpellCheckerPopUp.jsp"/>
```

Next an interface must be written in Javascript, RapidSpell will expect a type RSCustomInterface to be defined, of the following structure:

```
function RSCustomInterface(tbElementName){
      this.tbName = tbElementName;
      this.getText = getText;
      this.setText = setText;
      function getText(){
            //return the text from the text component named
this.tbName,
            //this may be HTML formatted text
            return .........
```

```
                }
                function setText(text){
                      //set the text in the text component to the text argument
                      //this may be HTML formatted text
                      ........
                }
          }
```

The two functions, getText and setText must be written to the specifications of the 3rd party component, note that if they will return XML(HTML) formatted text that the RapidSpellWebLauncher property ignoreXML should be set to true, otherwise the XML tags will be spell checked. In this example the interface will be interfacing with a regular HTML <textarea> in which the text contained is accessed using the format

```
document.formName.textAreaName.value
```

getText and setText can then be written as;

```
          function getText(){
                return eval('document.'+this.tbName+'.value');
          }
          function setText(text){
                eval('document.'+this.tbName+'.value = text') ;
          }
```

which dynamically accesses the text box named in the this.tbName variable. When RapidSpell runs it instantiates an object of type RSCustomInterface using the Javascript line

```
var interfaceObjectName = new RSCustomInterface(_textComponentName);
```

where _textComponentName is specified by the property TextComponentName in RapidSpellWebLauncher.

This gives the full code, for a custom interface as;

```
<%@ taglib uri="http://www.keyoti.com/" prefix="RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
   <HEAD>
      <script>
            function RSCustomInterface(tbElementName){
                  this.tbName = tbElementName;
                  this.getText = getText;
                  this.setText = setText;
                  function getText(){
                        return eval('document.'+this.tbName+'.value');
                  }
                  function setText(text){
                        eval('document.'+this.tbName+'.value = text') ;
                  }
            }
      </script>
   </HEAD>
```

```
   <BODY>
       <form action='exampleTextBox10-Custom.aspx' method='post' name='myForm'>
             <textarea name='tA'></textarea>
             <br>
             <RapidSpellWeb:rapidSpellWebLauncher id="rapidSpellWebLauncher"
                        textComponentName="myForm.tA"
                        mode="popup"
                        textComponentInterface="Custom"
                        ignoreXML="true"
                        rapidSpellWebPage="RapidSpellCheckerPopUp.jsp"/>
       </form>
</BODY>
</HTML>
```

Spell Checking Multiple Text Boxes

It is possible to check more than one text box on a Web Form, however with a couple of limitations, it is not possible to use the 'separate page' mode, only pop up may be used and also it is not possible to send the corrected text to a box other than the one it came from (the callBack property must not be set).

Code Example

Page containing the RapidSpellWebLauncher Tag

```
      <form action='exampleTextBox8-2boxes.jsp' method='post' name='myForm'>
            <input type='hidden' name='fMessage' value='complete'>
            <textarea name="sourceTextBox" wrap='true' cols='55' rows='10'></
textarea>
            <br>
            <RapidSpellWeb:rapidSpellWebLauncher
                              id="rapidSpellWebLauncher1"
                              rapidSpellWebPage="RapidSpellCheckerPopUp.jsp"
                              textComponentName="myForm.sourceTextBox"
                              />

            <textarea name="sourceTextBox2" wrap='true' cols='55' rows='10'></
textarea><br>
                         <RapidSpellWeb:rapidSpellWebLauncher
                              id="rapidSpellWebLauncher2"
                              rapidSpellWebPage="RapidSpellCheckerPopUp.jsp"
                              textComponentName="myForm.sourceTextBox2"
                              />
            </form>
```

Servlet Usage

RapidSpell Web provides convenient JavaBeans to enable use with Servlets. The Beans, RapidSpellWebLauncherBean and RapidSpellWebBean mirror the properties of the Tags described in this guide, and given the 'single round trip' nature of RapidSpell Web make adding spell checking to new and existing Servlets relatively easy.

Code Example Extract From RSServletUsage.zip Example App

Page containing the RapidSpellWebLauncher.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import com.keyoti.rapidSpell.web.*;

/** The servlet which will use the launcher button */
public class SpellUser extends HttpServlet{

      /**Process a GET request*/
      public void doGet(HttpServletRequest request,
                        HttpServletResponse response){
            processRequest(request, response);
      }

      /**Process a POST request*/
      public void doPost(HttpServletRequest request,
                        HttpServletResponse response){
            processRequest(request, response);
      }

      /** Generic request processor */
      protected void processRequest(HttpServletRequest request,
                              HttpServletResponse response){
            try{
                  PrintWriter pW = response.getWriter();
                  pW.println("<html><body><form name="f1"><textarea name='n1'
cols=50 rows=5></textarea>");

                  //Add RapidSpellWebLauncher to page
                  RapidSpellWebLauncherBean launcher = new
RapidSpellWebLauncherBean();
                  launcher.setTextComponentName("f1.n1");
                  launcher.setRapidSpellWebPage("/rsuser/RSWebPopUp");
                  launcher.writeHtml(pW, request);
```

```
                  pW.println("</form></body></html>");
            } catch (IOException e){
                  //................
            }
      }
}
```

Page containing RapidSpellWeb.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import com.keyoti.rapidSpell.web.*;


/** Note that if the user dictionary is being used this servlet will receive GET
requests aswell as the usual POST requests. */
public class RSWebPopUp extends HttpServlet{
      public void doGet(HttpServletRequest request,
                        HttpServletResponse response){
            processRequest(request, response);
      }
      public void doPost(HttpServletRequest request,
                        HttpServletResponse response){
            processRequest(request, response);
      }

      protected void processRequest(HttpServletRequest request,
HttpServletResponse response){
            try{
                  PrintWriter pW = response.getWriter();
                  pW.println("<html><body><center>");
                  //Add RapidSpellWeb to page
                  RapidSpellWebBean checker = new RapidSpellWebBean();
                  checker.writeHtml(pW, request);
                  pW.println("</center></body></html>");
            } catch (IOException e){
                  //...........
            }
      }
}
```

Please consult the Servlet zip file for a demonstration application. The same principle applies to the RapidSpellWInlineBean.

Spell Checking Multiple Text Boxes With One Button

It is possible to check more than one text box on a page with one button, using the RapidSpellWebMultiple Tag (or RapidSpellWebMultipleBean).

Code Example Extract From RSWL-MultipleBoxes.jsp

Page containing the RapidSpellWebMultiple.

```
<%@ taglib uri="http://www.keyoti.com/" prefix="RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
      <BODY >
             <form action='exampleTextBox1.jsp' method='post' name='myForm'>

                   <!---------------------- Text box 1 ------------------->
                   <textarea name="sourceTextBox" wrap='true' cols='55'
rows='10'>This is boxx 1.</textarea>

                   <br><br>
                              <RapidSpellWeb:rapidSpellWebLauncher
                              id="rapidSpellWebLauncher"
                              rapidSpellWebPage="RapidSpellCheckerPopUp.jsp"
                              textComponentName="myForm.sourceTextBox"
                              RSMultipleID="RapidSpellWebMultiple1"
                                    />
                  <br><br>
                   <!---------------------- Text box 2 ------------------->
                   <textarea name="sourceTextBox2" wrap='true' cols='55'
rows='10'>This is boxx 2.</textarea>
                   <br><br>
                               <RapidSpellWeb:rapidSpellWebLauncher
                              id="rapidSpellWebLauncher2"
                              rapidSpellWebPage="RapidSpellCheckerPopUp.jsp"
                              textComponentName="myForm.sourceTextBox2"
                              RSMultipleID="RapidSpellWebMultiple1"
                                    />
                  <br><Br>
                   <!---------------------- Text box 3 ------------------->
                   <textarea name="sourceTextBox3" wrap='true' cols='55'
rows='10'>This is boxnumber 3.</textarea>
                   <br><br>
                               <RapidSpellWeb:rapidSpellWebLauncher
                              id="rapidSpellWebLauncher3"
                              rapidSpellWebPage="RapidSpellCheckerPopUp.jsp"
```

```
                                textComponentName="myForm.sourceTextBox3"

                    RSMultipleID="RapidSpellWebMultiple1"
                                        />



<RapidSpellWeb:rapidSpellWebMultiple id="RapidSpellWebMultiple1" tabIndex="1"
rapidSpellWebLaunchers="rapidSpellWebLauncher,rapidSpellWebLauncher2,rapidSpellWebLauncher3"/
>

             </form>
             <p><font face='arial, helvetica'><a href="index.html">Back to
contents page.</a></font>
      </BODY>
</HTML>
```

Notice that the RapidSpellWebMultiple Tag refers to all the RapidSpellWebLaunchers that it should start in the course of the spell check. Also, each RapidSpellWebLauncher refers back to the RapidSpellWebMultiple Tag. All controls must have the "id" attribute set, in order for this to work. If a RapidSpellWebLauncher is not correctly referenced, it will still appear on the form as button, otherwise it will be hidden.

SSL Site Usage

RapidSpell Web is compatible with SSL web-sites. To use the components over SSL the SSLFriendly attribute must be set to true in the RapidSpellWeb and RapidSpellWebLauncher Tags/Beans, and a file named "blank.html" must be present in the same folder as the page where RapidSpellWeb and RapidSpellWebLauncher are used. The purpose of "blank.html" is to give the browser something to load in the Iframes using SSL. If this doesn't happen then a 'mixed content' warning will appear.; "blank.html" should be a bare bones html page.

Code Example SSL Usage

Page containing the RapidSpellWeb control.

```
<%@ taglib uri="http://www.keyoti.com/" prefix="RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html><body><Center>
<RapidSpellWeb:rapidSpellWeb SSLFriendly="true"/>
</center></body></html>
```

Blank.html

```
<html><body></body></html>
```

Modal Dialog Usage

When RapidSpell Web is used in popup mode the popup is by default non modal (user focus can go anywhere). To make the popup modal (user focus is limited to the spell checker) set the "modal" attribute in RapidSpellWebLauncher to true and copy RapidSpellModalHelper.html from the distribution demo folder to the folder where RapidSpellWebLauncher is used.

Examples (Inline/As-You-Type Spell Checker)

Basic Static Inline

When RapidSpellWInline is working with a plain textarea or input field, it works in 'static' mode. This means that underlines are shown as an overlay over the text box. This is the most compatible mode of operation and works with all modern browsers (Internet Explorer 5+, Netscape 6+, Firefox 1, Opera, and Safari on the PC and Mac). To work with a textbox the textComponentID attribute in RapidSpellWInline must be set to that of the text box.

Code Example From RSWI-Static.jsp

```
            <textarea id="tb1" cols="30" rows="5" style="font-family: Verdana,
Sans-Serif; font-size:11pt;">A regular plain textarea!</textarea>

            <RapidSpellWeb:rapidSpellWInline
            textComponentID="tb1"
            id="rswi1"
            />
```

Dynamic And As You Type Inline

To enable 'dynamic' (which means that the text will be editable AND underlined) checking, a RapidSpellWInlineTextBox must be used. To go further and enable 'as you type' checking, just set checkAsYouType="true" in RapidSpellWInline.

Code Example From RSWI-AsYouType.jsp

```
<form>

        <RapidSpellWeb:rapidSpellWInlineTextBox id="rswitb1"
            width="300" height="300"
            >I'm as yuu typed</RapidSpellWeb:rapidSpellWInlineTextBox>

        <RapidSpellWeb:rapidSpellWInline
            id="rswi1"
            textComponentID="rswitb1"
            checkAsYouType="true"
            showButton="false"
        />
</form>
```

Code Example Setting Style From RSWI-DynamicWithStyle.jsp

```
<RapidSpellWeb:rapidSpellWInlineTextBox id="rswitb1"
            width="300" height="300"
            styleFontFamily="Batang, Garamond, Serif"
```

```
            styleFontSize="14pt"
            styleBackColor="#ccffff"
            styleBorderStyle="dotted"
            styleBorderWidth="1"
            styleBorderColor="black"
            >This shows some of the style settings available with the dynamic
text box, and non as you type mode.</RapidSpellWeb:rapidSpellWInlineTextBox>

        <RapidSpellWeb:rapidSpellWInline
            id="rswi1"
            textComponentID="rswitb1"
            showButton="true"
        />
```

User Dictionaries (Add Option)

As with the RapidSpellWebLauncher tag, to activate the "Add" option just specify the userDictionaryFile attribute to a valid text file path on the server.

Code Example

```
       <RapidSpellWeb:rapidSpellWInline
            id="rswi"
            textComponentID="..."
            userDictionaryFile="c:/somedirectory/dictionary.txt"
        />
```

Multiple Textboxes

Regardless of whether using a plain (static) or dynamic textbox, setting up multiple RapidSpellWInline tags to work with one button simply requires the use of the RapidSpellWebMultiple tag. To do this, add RapidSpellWebMultiple to the page and give it an "id" attribute. Then set the rsMultipleID attribute in each RapidSpellWInline to the id of the multiple tag. Lastly, set rapidSpellWebLaunchers to a list of ids to launch.

Note; if activating As You Type, do not use RapidSpellWebMultiple as it is unnecessary and will confuse the behavior.

Code Example From RSWI-MultipleBoxes.jsp

```
        <textarea  id="ta" name="ta" rows=14 cols=30 >In thispage we
demenstrate</textarea>

        <RapidSpellWeb:rapidSpellWInline
            id="rswi"
            textComponentID="ta"
            rsMultipleID="rswm"
```

```
        />

        <textarea  id="ta2" name="ta2" rows=14 cols=30 >multiple textboxes being
checked</textarea>

        <RapidSpellWeb:rapidSpellWInline
            id="rswi2"
            textComponentID="ta2" rsMultipleID="rswm"
        />

        <textarea  id="ta3" name="ta3" rows=14 cols=30 >with one button.
        Note that RapidSpellWInlineTextBox could have been used in place of
&lt;textarea&gt;</textarea>

        <RapidSpellWeb:rapidSpellWInline
            id="rswi3"
            textComponentID="ta3" rsMultipleID="rswm"
        />

        <RapidSpellWeb:rapidSpellWebMultiple
rapidSpellWebLaunchers="rswi,rswi2,rswi3" id="rswm"
        showFinishedMessage="false" />
```

3rd Party Textboxes

RapidSpellWInline automatically identifies editable elements and therefore can work with most 3rd party components very simply. When the Automatic interface is unable to work with an editor, writing a Custom interface (as with RapidSpellWebLauncher) is an alternative option.

In this example, the IFRAME "HTB" is made editable and images are used in place of the button.

Code Example From RSWI-3rdParty.jsp

```
<RapidSpellWeb:rapidSpellWInline id="rswi"
textComponentID="HTB"
ignoreXML="true"
buttonImageMouseOut="images/out1.png"
buttonImageMouseOver="images/over1.png"
buttonImageMouseDown="images/down1.png"   />

<iframe id="HTB" width="400" height="300"></iframe>
```

Using With Struts

RapidSpellWInline can only work with textboxes that have the "id" attribute set in HTML. To have Struts output an "id" attribute, the "styleId" attribute must be set to some value in the JSP tag.

Eg.

```
<html:textarea property="title" styleId="tb12"/>
<RapidSpellWeb:rapidSpellWInline id="rswi1" textComponentID="tb12"/>
```

This works well, but when attempting to use with an Iterator tag, a runtime expression is needed to set the styleId, see below.

Code Example With Iterator

```
<%
            int i=0;
            String launchers = "";

            %>
            <logic:iterate  id="myCollectionElement" name="list2">
                <bean:write name="myCollectionElement" /><br />
                <%
                    i++;
                    String id="tb"+i;
                    String rswiID="r"+id;
                    launchers +=rswiID+",";
                %>
                Field: <html:textarea property="title" indexed="true"
                        styleId="<%=id%>"/>
                  <br />
                  <br />
                <RapidSpellWeb:rapidSpellWInline
                  id="<%=rswiID%>"
                  textComponentID="<%=id%>"
                   rsMultipleID="rswm"
                />
            </logic:iterate>

            <%//pull off trailing comma
            launchers=launchers.substring(0, launchers.length()-1);
            %>

            <RapidSpellWeb:rapidSpellWebMultiple id="rswm"
                  rapidSpellWebLaunchers="<%=launchers%>"
                   showFinishedMessage="false"
            />
```

Using With JSF

Use with JSF is mostly the same as with JSP tags, however care must be taken when setting the textComponentID attribute;

With RapidSpellWInlineTextBox - set textComponentID to the "id" of the text box

With regular text areas - set textComponentID to the "id" of the text box as rendered in HTML (typically "formID:textBoxID")

Code Example With RapidSpellWInlineTextBox

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

<jsp:root version="2.1" xmlns:f="http://java.sun.com/jsf/core" xmlns:h="http://java.sun.com/jsf/html"
xmlns:jsp="http://java.sun.com/JSP/Page" xmlns:webuijsf="http://www.sun.com/webui/webuijsf"
xmlns:ors="http://www.keyoti.com/"
>
  <jsp:directive.page contentType="text/html;charset=UTF-8" pageEncoding="UTF-8"/>
  <f:view>
    <webuijsf:page binding="#{Page1.page1}" id="page1">
      <webuijsf:html binding="#{Page1.html1}" id="html1">
        <webuijsf:head binding="#{Page1.head1}" id="head1">
          <webuijsf:link binding="#{Page1.link1}" id="link1" url="/resources/stylesheet.css"/>
          </webuijsf:head>
        <webuijsf:body binding="#{Page1.body1}" id="body1" style="-rave-layout: grid">
          <webuijsf:form binding="#{Page1.form1}" id="form1">

            <ors:rapidSpellWInlineTextBox id="rswitb"  width="300" height="300"/>
            <ors:rapidSpellWInline id="rswi" textComponentID="rswitb"/>

          </webuijsf:form>
        </webuijsf:body>
      </webuijsf:html>
    </webuijsf:page>
  </f:view>
</jsp:root>
```

Code Example With Standard TextBox

(as above example, except with this change)

```
<webuijsf:form binding="#{Page1.form1}" id="form1">

     <h:inputTextarea id="ta"/>
      <ors:rapidSpellWInline id="rswi" textComponentID="form1:ta"/>

</webuijsf:form>
```

Examples (Non-GUI Spell Checker)

The RapidSpellChecker Bean component is of particular use in non-GUI scenarios such as the web. Below is a very simple excerpt of how this component can be used.

Code Example

```
      .........
      RapidSpellChecker c = new RapidSpellChecker();
      BadWord badWord;
      Enumeration suggestions;
      //check some text.
      c.check("This is sume text.");
      //iterate through all bad words in the text.
      while((badWord = c.nextBadWord())!=null){
            System.out.println(badWord.getWord() + "- is not spelt correctly.
Suggestions:");
            try{
                  //get suggestions for the current bad word.
                  suggestions = c.findSuggestions().elements();
                  //display all suggestions.
                  while(suggestions.hasMoreElements()) {
                        System.out.println(suggestions.nextElement());
                  }
                  //change the bad word in the text with "replacement".
                  c.changeBadWord("replacement");
            } catch (NoCurrentBadWordException e){
                  System.err.println(e);
            }
      }
      System.out.println(c.getAmendedText());
      .........
```

Advanced Topics

License Keys In Multi Server Development Environments

It is common practice to use multiple servers in the development of web applications, for example, an application may go through a lifecycle;

developer machines ("localhost") -> single development test server ("devServer") -> QA server ("qaServer") -> production server ("prodServer")

The current licensing system requires keys for "devServer", "qaServer" and "prodServer" if they are accessed through non local browsers (note; 'devServer' and 'qaServer' keys will be provided free), this poses a problem since it is usually unacceptable to change the "licenseKey" attribute in the tag source code for each server.

Since each server has it's own environmental settings, it is likely that a config file is being used already, if the config file is unique to each server, then the RapidSpellWeb license key for each server can also be stored there. Once this is done it is easy to set the licenseKey attribute at runtime

Eg for RapidSpellWeb tag (same applies for RapidSpellWInlineHelper);

```
<%@ taglib uri="http://www.keyoti.com/" prefix="RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<title>Spell Check</title>
<body>
<Center>

<%
      String licenseKey = ....//Obtain license key from config file
%>

<RapidSpellWeb:rapidSpellWeb licenseKey="<%= licenseKey %>"/>

</center>
</body>
</html>
```

Suggestions Algorithm

It is possible to choose the algorithm used to find suggestions for spelling errors, there are two algorithms on offer, 'hashing' (default) and 'phonetic'. Each algorithm has it's own advantages and disadvantages;

Hashing (default): Very fast, roughly 40 times faster than the phonetic algorithm, works best on typographic errors and common spelling mistakes. Similar to most spell checker suggestions.

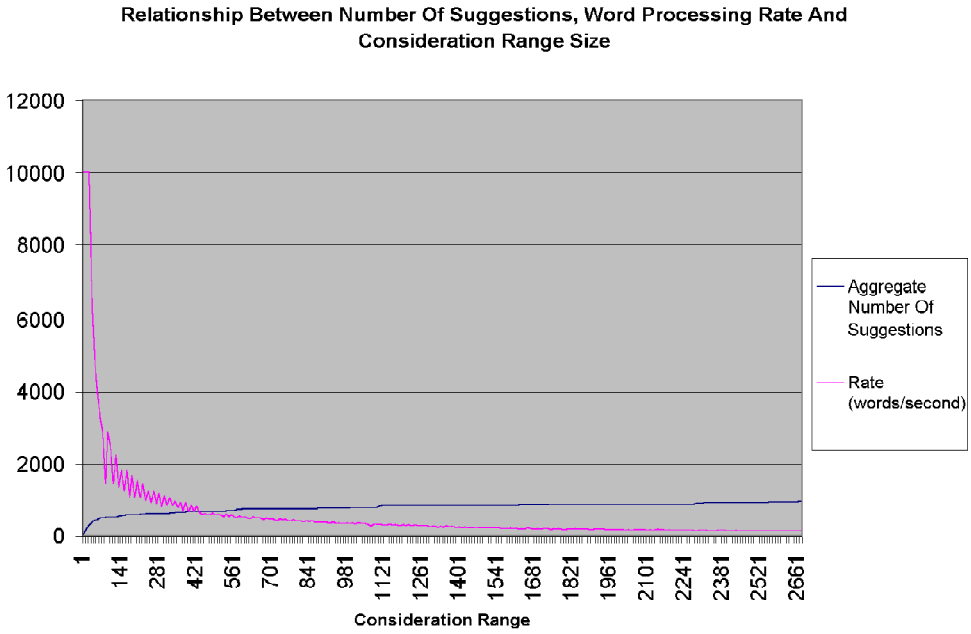
Phonetic (Sounds like, English only): Slower, works well with genuine attempts to spell correctly, works poorly with typographic errors. Can appear to produce wild results, although results do sound like the

word in question. Does not work with non English languages. Uncommon.

It is recommended that you choose the suggestions algorithm in line with your target audience. The phonetic algorithm, although appearing worse overall, is more suited for people who have not spoken English for very long, e.g. non native English speakers and children.

Consideration Range

The hashing suggestion algorithm is very fast because it accurately cuts down the list of words to look at as possible suggestions. The factory default consideration range is 80, which provides optimal speed with a loss of only a fraction of possible suggestions. It is possible to set the consideration range through the RapidSpellWebLauncher control and RapidSpellChecker class. The relationship between speed, number of suggestions and consideration range is shown below.



Client Side Events

It is possible to determine when spell checking has finished (in pop up mode) by using the FinishedListener property of RapidSpellWebLauncher. This property can be set to the name of a JavaScript (ECMAscript) function in the main form, which will be called from the pop up when the user finishes checking the document (either by completing the check, or by clicking the 'Finish' button). The function must not expect arguments and be with in the scope of the whole page.

For example;

```
..............
<script>
      function notifyDone(){
            alert('Finished event captured.');
      }
</script>
..............

 <RapidSpellWeb:rapidSpellWebLauncher
                        rapidSpellWebPage="RapidSpellCheckerPopUp.jsp"
                                    textComponentName="myForm.sourceTextBox"
      finishedListener="notifyDone"
      />
..............
```

This example is suited to web pages with single text boxes, for an example that works with multiple text boxes, please see the file "exampleTextBox11-MultipleCloseEvent.jsp".

Conclusion

Thank you for using the RapidSpell Web component, we hope that you experience much success with it. Keyoti is committed to improving all of it's products and truly values customer feedback of any kind. If you would like to contact us about any of our products please do so through our web site.

http://www.keyoti.com/contact.html

We thrive on suggestions for components, if you have a component wish please don't hesitate to get in touch, it may be the next thing we develop!