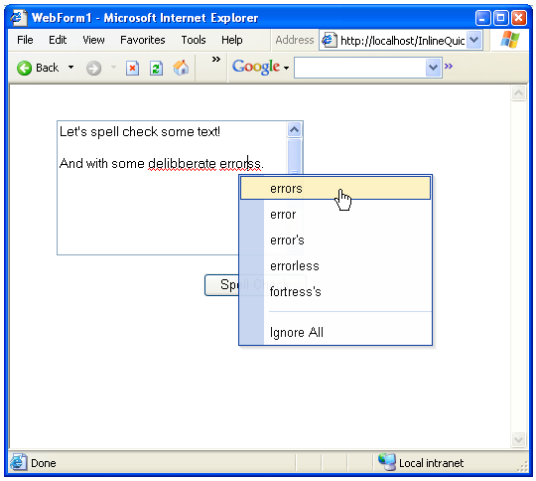
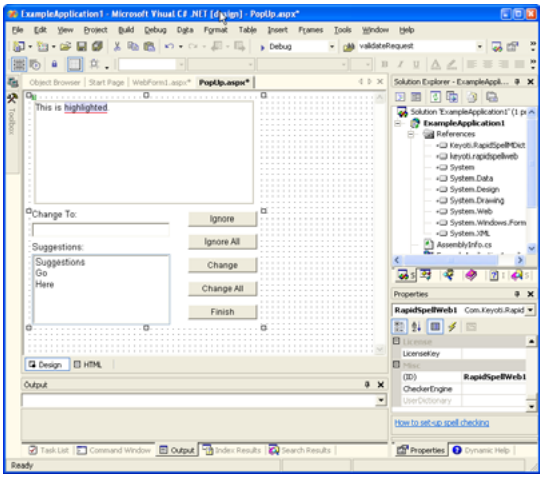
RapidSpell Web

v3.3.0 .NET 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 2003-2005, 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. Development and Distribution. If you purchased a One Developer License Version then you may copy and use the RapidSpell Web distribution file (.zip) on any systems you use but for your development use only. If you purchased a multi developer license version then the number of developers specified in the license type are afforded the rights of a One Developer License. Development use is defined as one person having the ability to read or copy any of the files originally contained in the RapidSpell Web distribution file including (but not limited to) the .dll, .csc, .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. 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. The 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 dll files as part of any .NET 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, 'Web Control' or 'Server Control'.

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

Overview

RapidSpell Web provides 6 spelling controls to add spell checker functionality to your web applications by simply adding the Web Control to an aspx page (Web Form) or directly accessing the classes in a code behind page. There are no programming skills required to use RapidSpell Web, making it accessible to ASP.NET page designers aswell as programmers. The UI provides all the usual features, add, change, change all, ignore, ignore all, and smart suggestions. It interactively checks ASP.NET text controls, any HTML text box component such as <textarea> and <input type='text'> aswell as 3rd party components such as Dart's PowerWEB TextBox. 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 Web Control component can check any HTML text component (e.g., <textarea>, <input type='text'> and 3rd Party) allowing for Internet/Intranet applications such as:

- Email
- Messaging
- Online utilities (web page generators, order placement)
- Data input

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

- Server applications
- Console based applications

Requirements

The components were built to .NETSDK 1.0/1.1/2.0 specification. The RapidSpell assembly files are ~4.5MB in size together. This product works in IE5+, Firefox1+, NS6+, Opera6+, Mozilla 1.3+ on the PC and IE5.5+, Firefox1+, and Safari 1.3+ on the Mac.

Improvements

Please see the ReleaseNotes.txt text file distributed with the product for a list of changes between versions.

The Dictionaries

Included are 3 dictionary assemblies, UK and US English dictionaries, each containing ~110K words, and a combined UK and US dictionary containing ~115K words. Other English dictionaries are available free at;
http://keyoti.com/products/rapidspell/dotnetweb/dictionaries/english-variations
These are included to give the developer choice, smaller dictionaries run faster, and are more accurate, but raise spelling error queries more frequently with unusual words.

Installation

To use RapidSpell Web, you must add the Keyoti.RapidSpellWeb.dll (for .NET 1) **or** Keyoti.RapidSpellWeb.ASP.NETv2.dll (for .NET 2) and Keyoti.RapidSpellMDict.dll files to the bin directory of your web application or server and reference the assembly Keyoti.RapidSpellWeb/ Keyoti.RapidSpellWeb.ASP.NETv2 in your aspx page.

For Visual Studio 2002/2003

Simply add the RapidSpell Web assembly to your Toolbox
1. Select the "Components" tab in the Toolbox slider

2. Right click in the Toolbox area and click "Customize Toolbox..."/"Add Remove Items"
3. Find the RapidSpell.... controls in the list and check them
5. Click Ok. the RapidSpellWeb, RapidSpellWInline, RapidSpellWInlineHelper, RapidSpellWInlineTextBox , RapidSpellWebMultiple and RapidSpellWebLauncher controls have been added to your Toolbox and can be dragged onto forms.

For Visual Studio 2005

Simply add the RapidSpell Web assembly to your Toolbox
1. Select the "Components" tab in the Toolbox slider
2. Right click in the Toolbox area and click "Choose Items"
3. Find the RapidSpell.... controls in the list and check them (ensure you select the controls in the Keyoti.RapidSpellWeb.ASP.NETv2 assembly!)
5. Click Ok. the RapidSpellWeb, RapidSpellWInline, RapidSpellWInlineHelper, RapidSpellWInlineTextBox , RapidSpellWebMultiple and RapidSpellWebLauncher controls have been added to your Toolbox and can be dragged onto forms.

For IIS (ASP.NET 1.x)

1. Copy Keyoti.RapidSpellWeb.dll and Keyoti.RapidSpellMDict.dll files in to the bin directory off your application root.
2. Add

```
<%@ Register TagPrefix="RapidSpellWeb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb" %>
```

to the top of the aspx page where you will use the RapidSpell Web Control.
It is vital that in development and deployment that you keep Keyoti.RapidSpellWeb.dll and Keyoti.RapidSpellMDict.dll in the same directory as each other.

For IIS (ASP.NET 2.0)

1. Copy Keyoti.RapidSpellWeb.ASP.NETv2.dll and Keyoti.RapidSpellMDict.dll files in to the bin directory off your application root.
2. Add

```
<%@ Register TagPrefix="RapidSpellWeb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb.ASP.NETv2" %>
```

to the top of the aspx page where you will use the RapidSpell Web Control.

It is vital that in development and deployment that you keep Keyoti.RapidSpellWeb.ASP.NETv2.dll and Keyoti.RapidSpellMDict.dll in the same directory as each other.

For Borland C# Builder

Adding RapidSpell Web Controls to the Toolbox in C# builder is practically the same as in VisualStudio. If you have difficulty please email support@keyoti.com.

For SharePoint

Under .NET 1.x the controls require client files in an external directory. By default this is /Keyoti_RapidSpell_Web_Common/ (see 'Manually' below) - however SharePoint's URL mapping may cause issues - therefore the files under Keyoti_RapidSpell_Web_Common need to be outside of SharePoint's control. RapidSpellWebLauncher, RapidSpellWeb, RapidSpellWInline and RapidSpellWInlineTextBox each have a ClientFilesFolder property which can be used to specify the absolute location of these files. By default it is /Keyoti_RapidSpell_Web_Common/

If URL mapping is causing problems you will see Javascript errors on the page holding any spell checker controls.

Deployment

Deployment of RapidSpell Web .NET onto a server can be done in one of the following ways.

Using The Product MSI Installer

If possible, run the MSI installer that was initially used to install the controls on the development machine. This will perform the steps below automatically. You can then either install Keyoti.RapidSpellWeb.dll (orKeyoti.RapidSpellWeb.ASP.NETv2.dll) and Keyoti.RapidSpellMDict.dll into the GAC using gacutil, or you can deploy these DLLs with your project's bin directory. See note below if installing in GAC.

Manually

Manual deployment is fairly simple;
1. [For ASP.NET 1.x users only] Copy the "Keyoti_RapidSpell_Web_Common" directory from the development machine (underneath where RapidSpell Web was installed) to the wwwroot folder on the server. This should result in URLs like http://servername/Keyoti_RapidSpell_Web_Common/ mapping to this directory. You can set it up elsewhere as a virtual directory if you prefer.
2. Either install Keyoti.RapidSpellWeb.dll (or Keyoti.RapidSpellWeb.ASP.NETv2.dll) and Keyoti.RapidSpellMDict.dll into the GAC using gacutil, or you can deploy these DLLs with your project's bin directory. If installing in the GAC please see Note below.

Note

As with any assemblies in the GAC, your aspx pages must register the assembly using a full reference. This means the register tag in your aspx pages should contain the correct version, culture and public key. Eg;

```
<%@ Register TagPrefix="RapidSpellWeb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb, Version=3.0.0.0, Culture=neutral,
PublicKeyToken=58d9fd2e9ec4dc0e" %>
```

otherwise the ASP.NET runtime will not be able to find the assembly in the GAC, and they will need to be in the project's bin dir.

License Keys

Upon purchase an 'activation key' would have been sent by email, please see the instructions in the file 'license-keys.txt' which outlines the use of activation keys, and the generation of 'deployment keys'.

Once you have generated deployment keys at http://www.keyoti.com/KeyMaker/main.aspx you should set the LicenseKey property of RapidSpellWeb or RapidSpellWInlineHelper Control, or in your web.config (see below). If the deployment keys are not correct then attempts to use the control on a remote server will result in alert messages.

Eg. LicenseKey property

```
<RapidSpellWeb:RapidSpellWeb id="rapidSpellWeb" runat="server"
   LicenseKey="12345678901234567890123456789012345678901234567890"
   />
```

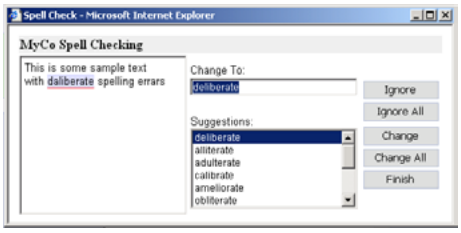
Eg. web.config;

```
<appSettings><add key="Keyoti-RapidSpellWeb-LicenseKey" value="..."/></appSettings>
```

if you have more than one domain/servername (and therefore deployment key) for the app. you can easily specify a second key by adding a key called "Keyoti-RapidSpellWeb-LicenseKey2".

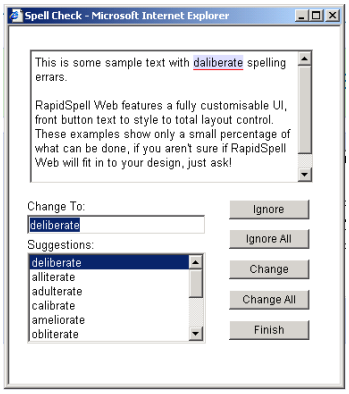
If you are using a multi-server development environment (eg. dev-server -> test-server -> production-server) please see the 'Advanced' section for license key handling strategy.

The Components

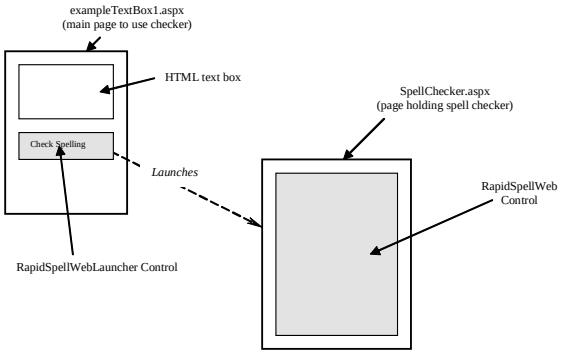


Dialog Spell Checking

The RapidSpell Web dialog based checker is actually used via two Web Controls, called RapidSpellWebLauncher and RapidSpellWeb, which launch the spell check page and show the spell checker, respectively. RapidSpell Web also includes RapidSpellWebMultiple, used for checking multiple text boxes.



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.



RapidSpellWeb Control runs through the entirety of the HTML text box specified in the RapidSpellWebLauncher Control 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 or, if in popup mode the popup will close and the text will be pasted into the text area specified in CallBack (or if not specified into the original text area).

RapidSpellWeb Summary

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

Incorrectly spelt words are highlighted in the preview pane as the user is queried.
Fully customizable user-dictionary at development and install time.
Fully customizable UI layout, style and text.
Undo, Ignore, Ignore All, Add, Change, Change All functionality.

RapidSpellWebLauncher Summary

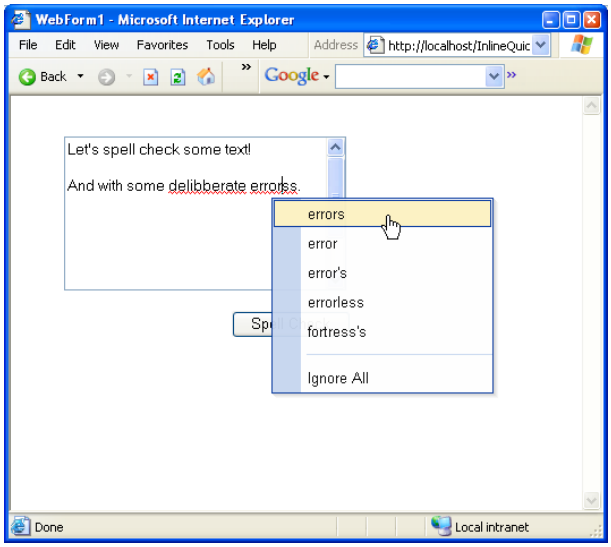
This control should be integrated in to the form containing the text area to be checked, it is rendered visibly as a button but contains hidden form fields and Javascript to open the page containing the RapidSpellWeb control, this control's properties are used to define the behaviour of the spell checker, 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.
Image button mode with 3 mouse states

RapidSpellWebMultiple Summary

This control makes checking multiple text boxes with one button very simple. It contains most of the properties of RapidSpellWebLauncher, and when rendered on a page, appears as a button or image button. The significant property is RapidSpellWebLaunchers, which should equal a comma delimted list of RapidSpellWebLaucher/RapidSpellWInline IDs (each RapidSpellWebLauncher/RapidSpellWInline is configured to spell check a text box, but is hidden at run-time). Please see the "Spell Checking Multiple Text Boxes With One Button" section of this guide for more info. When dropped onto a design surface, integrated help links appear under the property editor., which provide quick start help.

Inline Spell Checking



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 plugins or other client side technologies, however this control does provide user friendly in-document highlighting, without plugins or other special settings. There are 2 core components to this type of checking; RapidSpellWInline control 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. RapidSpellWInline uses the AJAX style of client programming to access the server, it does this in part with the RapidSpellWInlineHelper control, which must sit on a page by itself, although this control doesn't render as anything, it does provide the necessary Javascript for the RapidSpellWInline to function.

Besides the 2 core components there is also the RapidSpellWInlineTextBox control, which is an optional replacement to the standard ASP.NET TextBox control. RapidSpellWInlineTextBox fucntions almost identically to the standard ASP.NET 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 ASP.NET text box then it will cause the text box to become 'static' while the errors are highlighted.

RapidSpellWInline Summary

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 Summary

Is placed on an empty page and performs the actual spell checking for the RapidSpellWInline control.

Behaviour of spell checker can be changed through this control.

RapidSpellWInlineTextBox Summary

Is an OPTIONAL replacement for the standard ASP.NET 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).

Subclass of System.Web.UI.WebControls.TextBox

RapidSpellChecker

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 70,000 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.

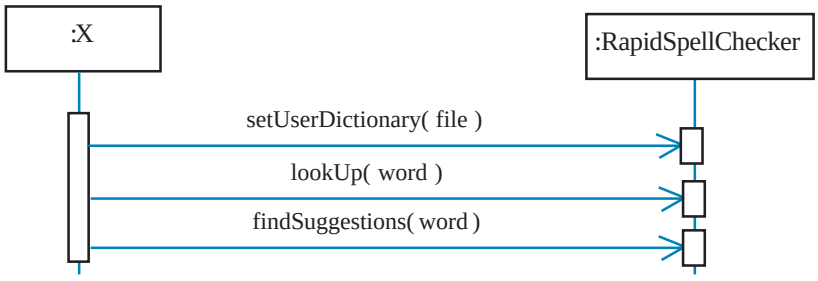
Alternatively RapidSpellChecker may be used in a simple query by query basis, calls to LookUp(word) and FindSuggestions(word) methods allow bare bones spell checking functionality.

Key API Points

Key Methods

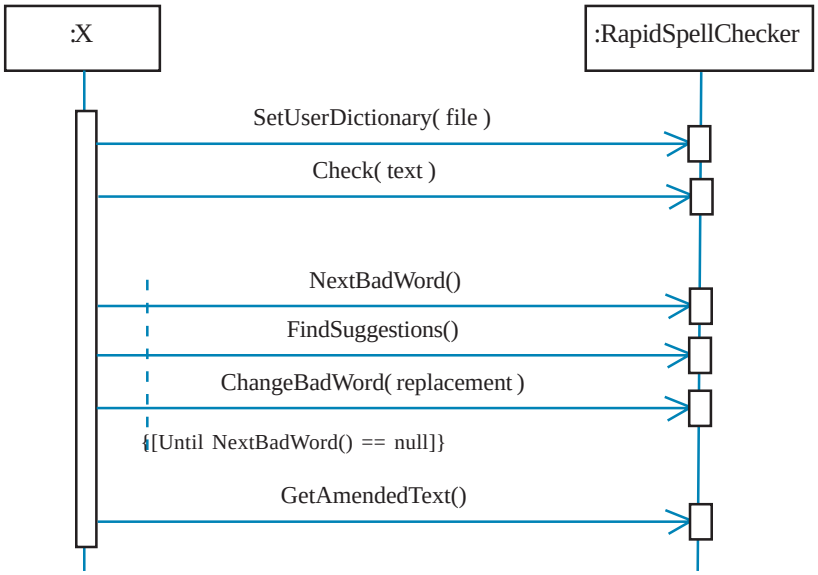
RapidSpellChecker()	-argumentless constructor
Check(text [, startPosition])	-begins checking text
FindSuggestions(word)	-finds suggestions for word
LookUp(word)	-checks if word is in either main or user dictionaries
SetUserDictionary(file)	-allows you to define which file will be used as a user dictionary
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 use cases section for examples.



Dict Files

RapidSpell has 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.DLL format will continue to be supported, the advantages and disadvantages of the formats are outlined below:

Dict File;

Advantages
- Smaller file size
- Swappable during run-time (necessary for other languages)
- Customizable by developer

Disadvantages
- Not cacheable in GAC

Existing MDict.DLL format;

Advantages
- Cacheable in GAC

Disadvantages
- Not runtime swappable
- Not customizable

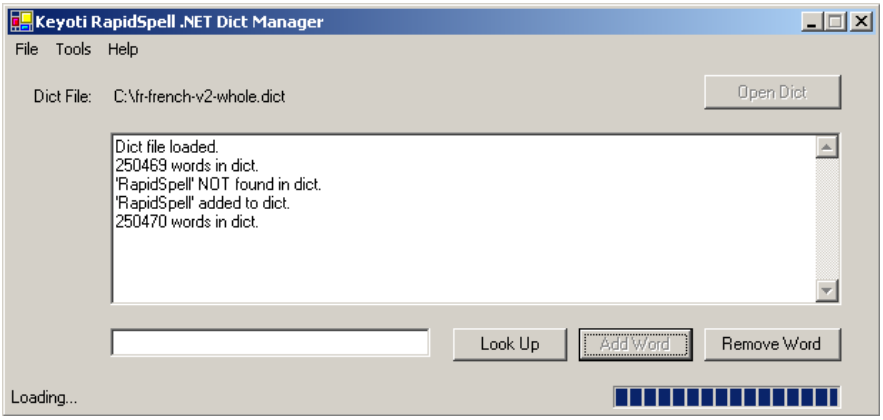
If it is desirable to cache the dictionary in the GAC then the MDict.DLL format should be used, however for all other uses the new Dict File format is preferable.

Using Dict Files

The older MDict.DLL type file must reside in the same directory as the main RapidSpell DLL, it will be used if the DictFile property of RapidSpellWebLauncher/RapidSpellWInline is null (C#) or Nothing (VB.NET). To use a Dict file simply set the DictFile property to the file path of the Dict file to be used on the server.

Dictionary Customization

It is possible to customize the dictionaries and create new ones using the convenient Dict Manager tool included in RapidSpell Web, please load the tool and consult the help system provided in it for more details.



The Dict Manager tool included in RapidSpell Web

Use Cases

Below are some use scenarios of the 6 spell checker components supplied, please refer to the API documentation for additional detail. These examples are from C# pages, however because they are tag based most of them are the same for VB.NET (please see the example Visual Studio projects for more code).

Simple Popup Spell Checking

To use RapidSpell Web to spell check a web page in popup mode, create a page holding the RapidSpellWeb Control, then add the RapidSpellWebLauncher Control 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

Page containing the RapidSpellWeb control

```
<%@ Page Language="C#" %>
<%@ Register TagPrefix="RapidSpellWeb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html><head><title>Spell Check</title></head>
<body>
<center>
<form runat=server id="Form1" method="post">
      <RapidSpellWeb:RapidSpellWeb id="rapidSpellWeb" runat="server"/>
</form>
</center>
</body>
</html>
```

Code Example

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

```
<%@ Page Language="C#"%>
```

```
<%@ Register TagPrefix="RapidSpellWeb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<BODY >
       <form runat=server id=Form1 method=post>
            <asp:textbox id="TextBox1" runat="server" Font-Size="9pt" Font-
Names="Arial" Height="220px" Width="469px" TextMode="MultiLine" Columns="60"
Rows="14"></asp:textbox>
           <rapidspellweb:rapidspellweblauncher id="RapidSpellWebLauncher1"
runat="server" RapidSpellWebPage="PopUp.aspx" TextComponentName="TextBox1" >
</rapidspellweb:rapidspellweblauncher>
       </form>
</body>
</HTML>
```

The main page uses the control RapidSpellWebLauncher 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). RapidSpellWebPage can be a relative or absolute URL.

Simple Inline Spell Checking

To use RapidSpellWInline to spell check a text box; add the control to the form with the text box, set the TextComponentID property to the ID of the text box, and set the RapidSpellWInlineHelperPage property to the URL of a page holding the RapidSpellWInlineHelper control (this page needs nothing other than the RapidSpellWInlineHelper control on it).

Code Example

```
<%@ Register TagPrefix="rapidspellweb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb" %>
<%@ Page language="c#"  AutoEventWireup="false"  %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
      <body MS_POSITIONING="GridLayout">
            <form id="Form1" method="post" runat="server">
                  <asp:TextBox id="TextBox1" style="Z-INDEX: 101; LEFT: 135px;
POSITION: absolute; TOP: 32px" runat="server"   Width="330px"
TextMode="MultiLine" Height="162px"></asp:TextBox>
```

```
      <RapidSpellWeb:RapidSpellWInline id="RapidSpellWInline1" style="Z-INDEX:
102; LEFT: 366px; POSITION: absolute; TOP: 210px" runat="server"
TextComponentID="TextBox1" RapidSpellWInlineHelperPage="rswihelper.aspx">
<Button BorderWidth="" BorderColor="" ForeColor="" BackColor="" Height=""
Width="" CssClass="" type="button" value="Spell Check"
ID="RapidSpellWInline1__ctl0"></Button>
      </RapidSpellWeb:RapidSpellWInline>
            </form>
      </body>
</HTML>
```

Code Example (Helper page)

The code above requires a page holding RapidSpellWInlineHelper, create one called rswihelper.aspx containing the following;

```
<%@ Page language="c#" AutoEventWireup="false" validateRequest="false"%>
<%@ Register TagPrefix="rapidspellweb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
      <body>
            <form id="Form1" method="post" runat="server">
            <RapidSpellWeb:RapidSpellWInlineHelper id="rswi1" runat="server"/>
            </form>
      </body>
</HTML>
```

Setting PopUp Style

It is very simple to define the style properties of elements as with any other .NET Web Control, eg;

Code Snippet Button Customization

```
<IgnoreButton
      Style="font-family:'Tahoma,Arial,Helvetica';font-size:10pt; border:1px
solid #b5bed6; background-color:#dddddd; width: 90px;"

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

```
onMouseOut="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. An alternative to setting the Style property is the CssClass property which can be used to specify a CSS class for the element.

Separate Page Spell Checking

To use RapidSpell Web to spell check a web page in separate mode, create a page holding the RapidSpellWeb control, then add the RapidSpellWebLauncher control 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.aspx

Page containing the RapidSpellWeb control.

```
<%@ Page Language="C#"%>
<%@ Register TagPrefix="RapidSpellWeb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb" %>
<HTML>
  <BODY >
<RapidSpellWeb:RapidSpellWeb id="rapidSpellWeb" runat="server">
      <IgnoreButton
            Style="font-family:'Tahoma,Arial,Helvetica';font-size:10pt;
border:1px solid #b5bed6; background-color:#dddddd; width: 90px;"

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

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

      />
      <IgnoreAllButton
            ....  />
      <AddButton
            ....  />
      <ChangeButton
            ....  />
      <ChangeAllButton
```

```
            ....  />
      <FinishButton
            ....  />
      </RapidSpellWeb:RapidSpellWeb>
</BODY>
</HTML>
```

Code Example From exampleTextBox5-Separate.aspx

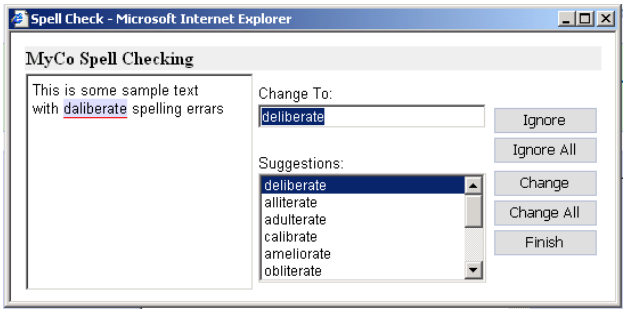
Page containing the RapidSpellWebLauncher control

```
<%@ Page Language="C#" %>
<%@ Register TagPrefix="RapidSpellWeb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
  <BODY>

      //if this page is to show the text input box
      <% if(Request.Form.Get("fMessage") == null){  %>
      <form action='exampleTextBox1.aspx' method='post' name='myForm'>
      <input type='hidden' name='fMessage' value='complete'>
      //if showing the text box for the first time, no spell check performed

      <% if(Request.Form.Get("RSCallBack") == null){   %>

            <textarea name="sourceTextBox" wrap='true' cols='40' rows='10'>
                  This is some sample text with daliberate spelling errars
            </textarea>

      <% } else { %>

            //spell check has been run, so show the corrected text
           <textarea name="sourceTextBox" wrap='true' cols='55' rows='10'>
                  <%= Request.Form.Get("CorrectedText") %>
            </textarea>

      <% } %>
          <br>

            <RapidSpellWeb:RapidSpellWebLauncher id="rapidSpellWebLauncherSep"
                  runat="server"
                  TextComponentName="myForm.sourceTextBox"
                  CallBack="exampleTextBox5-separate.aspx"
                  Mode="separate"
```

```
                    RapidSpellWebPage="SpellCheckSeparatePage.aspx" >
                    <Button Style="font-family:'Tahoma,Arial'; border:1px solid
#b5bed6; background-color:#dddddd;"

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

onMouseOut="this.style.backgroundColor='#dddddd';this.style.borderColor='#b5bed6';"
                          Value="Check Spelling Separate Page"
                    />
                </RapidSpellWeb:RapidSpellWebLauncher>
      <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.Form.Get("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.Html 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/>, <UndoButton>, <FieldDisplayLabel/>, <ChangeButton/>, <ChangeAllButton/>, <FinishButton/>, <ChangeToLabel/>, <ChangeToBox/>*, <SuggestionsLabel/>, <SuggestionsBox/>* - tags marked with * are required for correct functionality.

Code Example With Customized Layout

Page containing the RapidSpellWeb control, note because this has different dimensions you should modify the WindowWidth and WindowHeight properties in the RapidSpellWebLauncher that launches this, to 530 and 250 respectively.

```
<%@ Page Language="C#"%>
<%@ Register TagPrefix="RapidSpellWeb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<body>
<Center>
<RapidSpellWeb:RapidSpellWeb id="rapidSpellWeb" runat="server">
      <ChangeToBox Style="font-size:9pt; width:200px"/>
      <SuggestionsBox Style="font-size:9pt; width:200px"/>
      <PreviewPaneStyle Width="200" Height="190"/>
      <Layout Html="
            <table border=0>
            <tr><td colspan=3 bgcolor='#eeeeee'><b>MyCo Spell Checking</b>
            </td></tr><tr><td>
            <PreviewPane/>
            </td><td>
            <ChangeToLabel/>
            <br>
            <ChangeToBox/>
            <br>
            <SuggestionsLabel/>
            <br>
            <SuggestionsBox/>
            </td><td>
            <table border=0><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>
      "/>
</RapidSpellWeb:RapidSpellWeb>
</center>
</body>
</html>
```

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

Specifying User Dictionaries

Use of user dictionaries with RapidSpell Web can be 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 path is specified in the UserDictionaryFile property of the RapidSpellWebLauncher/RapidSpellWInline control. This property specifies the file path to the user dictionary file on the server, and since this property can be specified simply and dynamically using a 'DataBind', the developer has total control over which dictionary file is used for a user. Therefore it is upto the developer to manage a user's session and retreive the path to a particular user's/group's dictionary file, if required.

User dictionaries are simple text files, they contain a list of words, one per line, and are written to when the user clicks the 'Add' button. If the user dictionary file doesn't exist, it is created. If the user dictionary file cannot be accessed (due to an invalid path, or insufficient file access permission) then the 'Add' button is not displayed.

User Dictionary Strategy

It is often desirable to give each user their own user dictionary - in order to implement this it is clear that the application must use some sort of user session management (otherwise it is impossible to tell users apart). Given this general requirement it is simple to maintain separate user dictionaries for each user. By creating a directory for user dictionaries and making each dictionary file name dependant on a unique user

identifier (perhaps their username, or a database key) each user is given their own file, and at run-time the UserDictionaryFile property can be set accordingly (using a DataBind, or accessing the property in code-behind).

Eg. your application has a user with username "joe_user" (for simplicity we will assume that user names cannot contain characters that are illegal in filenames). A directory off the application directory called "user-dictionaries" is created, the ASP_NET process has permission to write to it. All that is necessary is to set the UserDictionaryFile property;

[Code-behind]

RapidSpellWebLauncher.UserDictionaryFile = MapPath("user-dictionaries/" + userName + ".txt")

or

RapidSpellWInline.UserDictionaryFile = MapPath("user-dictionaries/" + userName + ".txt")

Code Example Using DataBind

Page containing the RapidSpellWebLauncher control

```
<%@ Register TagPrefix="RapidSpellWeb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb" %>
<%@ Page Language="C#" Debug="true"%>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
      <HEAD>
      <script language="C#" runat="server">
      //set the user dictionary file, this path could be obtained
      //dynamically if the user has their own file.
      String userDicFile = "C:\\Inetpub\\wwwroot\\rapidSpellWeb\\userDict.txt";

      //on page load, bind the data
      void Page_Load(Object sender, EventArgs e) {
            Page.DataBind();
      }
            </script>
      </HEAD>
      <BODY>
            <form runat="server" method='post' id="form1">
                  <textarea id="sourceTextBox" name="sourceTextBox" wrap='true'
cols='40' rows='10'>This is some sample text with daliberate spelling errars</
textarea>
                  <br>
                  <RapidSpellWeb:RapidSpellWebLauncher
                  id="rapidSpellWebLauncher" runat="server"
```

```
                TextComponentName="sourceTextBox"
                Mode="popup"
                UserDictionaryFile="<%# userDicFile %>"
                RapidSpellWebPage="PopUp.aspx"
                />

        </form>
    </BODY>
</HTML>
```

In this example the user dictionary is specified in the string userDicFile which is passed to the UserDictionaryFile property of the RapidSpellWeb control using the <%# .... %> tags. The Page.DataBind() method is called when the page loads, which binds the value of userDicFile to the property UserDictionaryFile. The string was specified statically, but it could easily have come from a database, or session control object.

Using RapidSpell Web With 3rd Party Controls (Html Text Boxes)

3rd Party Controls can be used with RapidSpell Web, to use with a rich/html text box simply set IgnoreXML to true (this will mean HTML tags are ignored), set TextComponentInterface to the name of your Control (if not available the CustomInterface method should be used) and set TextComponentName to the ID of the Control. For example if you use Dart's PowerWEB TextBox and your web form code is;

```
<form id="Form1" name="Form1".....>
      .........
      <cc1:HtmlBox id="HtmlBox1" runat="server"></cc1:HtmlBox>
      .........
      .........
</form>
```

then you should set TextComponentName to "HtmlBox1" (the ID of the Control) and TextComponentInterface should be set to "PowerWEBTextBox". If your Html text box control isn't available in TextComponentInterface then you should contact the vendor and ask about RapidSpell Web support - it is possible to build custom support yourself, fairly simply (see below).

Code Example Using Html Text Boxes

Page containing the RapidSpellWebLauncher control

```
.....................
<form action='exampleTextBox6-3rdParty.aspx' method='post' name='myForm'
id='myForm' runat='server'>
       <cc1:HtmlBox id="HtmlBox1" runat="server"></cc1:HtmlBox>
       <RapidSpellWeb:RapidSpellWebLauncher id="rapidSpellWebLauncher"
runat="server"
            TextComponentName="HtmlBox1"
            Mode="popup"
            TextComponentInterface="PowerWEBTextBox"
            IgnoreXML="true"
            RapidSpellWebPage="RapidSpellCheckerPopUp.aspx">
       </RapidSpellWeb:RapidSpellWebLauncher>
</form>
.....................
```

To use RapidSpell Web with indirectly supported 3rd party controls 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.

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" runat="server"
                        TextComponentName="tA"
                        Mode="popup"
                        TextComponentInterface="Custom"
                        RapidSpellWebPage="RapidSpellCheckerPopUp.aspx">
```

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.form1.'+this.tbName+'.value');
          }
          function setText(text){
                eval('document.form1.'+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.

For Html text boxes the Javascript text access method (how you get the innerHTML out of the text box) is dependant on the vendor, however it is usually easy to identify - Html text boxes tend to work by setting the 'design mode' to true on either, Iframe body tags, div tags or span tags. To identify your vendors method, run a 'view-source' in the browser, and look for an Iframe, div or span, and see if design mode is set on for the element. For Html text boxes the inner Html (which is what is required by getText and setText), is accessed through the innerHTML property, eg; document.all['mySpanTag'].innerHTML.

The full code, for a simple text box custom interface is;

```
<%@ Page Language="C#" Debug="true"%>
<%@ Register TagPrefix="RapidSpellWeb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.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.myForm.'+this.tbName+'.value');
                }
                function setText(text){
                      eval('document.myForm.'+this.tbName+'.value = text') ;
                }
          }
      </script>
   </HEAD>
   <BODY>
       <form runat=server>
             <textarea ID='tA'></textarea>
             <br>
             <RapidSpellWeb:RapidSpellWebLauncher id="rapidSpellWebLauncher"
runat="server"
                        TextComponentName="tA"
                        Mode="popup"
                        TextComponentInterface="Custom"
                        IgnoreXML="true"
                        RapidSpellWebPage="RapidSpellCheckerPopUp.aspx">
             </RapidSpellWeb:RapidSpellWebLauncher>
       </form>
</BODY>
</HTML>
```

Please note, if you are using .NET 1.1 or higher you will need to add validateRequest='false' to the page directive in the page holding the RapidSpellWeb control (the popup) and the RapidSpellWInlineHelper control (for inline). Failure to do this will result in errors when spell checking text containing HTML.

Spell Checking Multiple Text Boxes With One Button

It is possible to check more than one text box on a Web Form with one button, using the RapidSpellWebMultiple control.

Code Example Multiple Text Boxes, One Button

Page containing the RapidSpellWebLauncher and RapidSpellWebMultiple controls

```
<%@ Page language="c#" AutoEventWireup="false"  %>
<%@ Register TagPrefix="rapidspellweb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
      <body MS_POSITIONING="GridLayout">
            <form id="MultipleTextBoxes" method="post" runat="server">

                  <asp:TextBox id="TextBox1" style="Z-INDEX: 100; LEFT: 21px;
POSITION: absolute; TOP: 65px" runat="server" TextMode="MultiLine" Width="404px"
Height="127px" Columns="40" Rows="4">This example demenstrates</asp:TextBox>

                  <asp:Label id="Label1" style="Z-INDEX: 109; LEFT: 21px;
POSITION: absolute; TOP: 31px" runat="server" Font-Names="Arial">Multiple text
box checking with one button</asp:Label>

                  <asp:TextBox id="TextBox2" style="Z-INDEX: 101; LEFT: 21px;
POSITION: absolute; TOP: 201px" runat="server" TextMode="MultiLine" Width="404px"
Height="127px" Columns="40" Rows="4">multiple textbox checking</asp:TextBox>

                  <asp:TextBox id="TextBox3" style="Z-INDEX: 103; LEFT: 21px;
POSITION: absolute; TOP: 335px" runat="server" Width="259px" Columns="40">Any
combination of text boexs on a form</asp:TextBox>

                  <rapidspellweb:RapidSpellWebLauncher
id="RapidSpellWebLauncher2" style="Z-INDEX: 104; LEFT: 435px; POSITION: absolute;
TOP: 213px" runat="server" TextComponentName="TextBox2"
RapidSpellWebPage="PopUp.aspx" CreatePopUpWindow="False"
PopUpWindowName="RapidSpellWebMultiple1_PopUpWin" ShowFinishedMessage="False"
ShowNoErrorsMessage="False" FinishedListener="RapidSpellWebMultiple1_NotifyDone"
ShowButton="False">
                  </rapidspellweb:RapidSpellWebLauncher>
```

```
                <rapidspellweb:RapidSpellWebLauncher
id="RapidSpellWebLauncher3" style="Z-INDEX: 105; LEFT: 294px; POSITION: absolute;
TOP: 339px" runat="server"  TextComponentName="TextBox3"
RapidSpellWebPage="PopUp.aspx" CreatePopUpWindow="False"
PopUpWindowName="RapidSpellWebMultiple1_PopUpWin" ShowFinishedMessage="False"
ShowNoErrorsMessage="False"  FinishedListener="RapidSpellWebMultiple1_NotifyDone"
ShowButton="False">
                </rapidspellweb:RapidSpellWebLauncher>

                <rapidspellweb:RapidSpellWebLauncher
id="RapidSpellWebLauncher1" style="Z-INDEX: 107; LEFT: 438px; POSITION: absolute;
TOP: 75px" runat="server"  TextComponentName="TextBox1"
RapidSpellWebPage="PopUp.aspx" CreatePopUpWindow="False"
PopUpWindowName="RapidSpellWebMultiple1_PopUpWin" ShowFinishedMessage="False"
ShowNoErrorsMessage="False" FinishedListener="RapidSpellWebMultiple1_NotifyDone"
ShowButton="False">
                </rapidspellweb:RapidSpellWebLauncher>

                <rapidspellweb:RapidSpellWebMultiple
id="RapidSpellWebMultiple1" style="Z-INDEX: 106; LEFT: 21px; POSITION: absolute;
TOP: 369px" runat="server"
RapidSpellWebLaunchers="RapidSpellWebLauncher1,RapidSpellWebLauncher2,RapidSpellWebLauncher3">
                </rapidspellweb:RapidSpellWebMultiple>

            </form>
        </body>
</HTML>
```

This code functions by hiding the buttons usually displayed by the RapidSpellWebLauncher control. The RapidSpellWebMultiple control is used to link all the RapidSpellWebLauncher/RapidSpellWInline controls, it renders as a button (or image) on the page and controls the firing of the hidden RapidSpellWebLauncher/RapidSpellWInline controls. The RapidSpellWebLaunchers property in RapidSpellWebMultiple is set to the IDs of the RapidSpellWebLauncher/RapidSpellWInline controls, comma delimted.

RapidSpellWebLauncher In A User Control

It may be desirable to package RapidSpellWebLauncher and other controls together into a single User Control, this makes component reuse even easier. For example a User Control could be created that contains RapidSpell Web and a text box, which would form a logical development unit.

Code Example User Control

UserControl.ascx (the user control)

```
<%@ Control Language="c#" AutoEventWireup="true" TargetSchema="http://
schemas.microsoft.com/intellisense/ie5"%>
<%@ Register TagPrefix="RapidSpellWeb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb" %>

<TABLE id="Table1" style="WIDTH: 499px; HEIGHT: 229px" cellSpacing="1"
cellPadding="2" width="499" align="left" border="0">
 <tr>
  <td colSpan="2">
<asp:TextBox id="TextBox1" runat="server" TextMode="MultiLine" Height="171px"
Width="363px" Columns="40" Rows="4"></asp:TextBox>
<br>
<RapidSpellWeb:RapidSpellWebLauncher id="RapidSpellWebLauncher1" runat="server"
TextComponentName="TextBox1"  RapidSpellWebPage="PopUp.aspx">
</RapidSpellWeb:RapidSpellWebLauncher>
 </td></tr>
</TABLE>
```

This user control is now easily used on a regular aspx form.

```
<%@ Page Language="C#" %>
<%@ Register TagPrefix="Acme" TagName="SpellBox" Src="UserControl.ascx" %>

<html>
<body style="font: 10pt verdana">

  <h3>Html Text Box Spelling Control page</h3>
  <form id="frmMenu" runat=server>
        <Acme:SpellBox runat="server"/>
  </form>

</body>
</html>
```

The result is a form unit that is easily reusable.

SSL Usage

RapidSpell Web is compatible with SSL web-sites. To use the controls over SSL the SSLFriendly property must be set to true in the RapidSpellWeb, RapidSpellWebLauncher and RapidSpellWInline controls - this requires a file called 'blank.html' in the Keyoti_RapidSpell_Web_Common virtual directory.

Code Example SSL Usage

Page containing the RapidSpellWeb control.

```
<%@ Page Language="C#" debug="false" validateRequest=false %>
<%@ Register TagPrefix="RapidSpellWeb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb" %>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
<title>Spell Check</title>
<body>

<RapidSpellWeb:RapidSpellWeb id="rapidSpellWeb" runat="server" SSLFriendly=true/>

</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 property in RapidSpellWebLauncher to true.

PostBack To Server On Spell Check Finish

RapidSpellWebLauncher and RapidSpellWebMultiple can post-back to the server when the spell check has completed, making it possible to read the updated text box on the server after corrections have been made. To enable this behavior set the PostBackOnFinish property to true; It is then appropriate to set a server-side listener:

Setting Server Side Finish Event in C#

```
RapidSpellWebMultiple1.Finish += new EventHandler(RapidSpellWebMultiple1_Finish);
....
private void RapidSpellWebMultiple1_Finish(object sender, System.EventArgs e)
{
      ....
}
```

Setting Server Side Finish Event in VB.NET

```
 Private Sub SpellCheckFinished(ByVal sender As Object, ByVal e As EventArgs)
Handles RapidSpellWebMultiple1.Finish
      ....
End Sub
```

The RapidSpellWebLauncher control will post back to the server whenever spell check finishes (either through completion or cancellation), but will NOT post back when used in conjunction with RapidSpellWebMultiple.

The RapidSpellWebMultiple control will post back to the server when checking of all text boxes has been completed or checking has been cancelled.

Spell Checking DataGrid/List Controls

This is as simple as multiple text box checking. To begin with place a RapidSpellWebLauncher or RapidSpellWInline control in the item template or edit item template (where ever the text box is). Configure the Control's properties as usual. If the spell checker is in the edit item template then you're done, if however the spell checker is in the item template then you probably want to use RapidSpellWebMultiple to check all textboxes at once. To do this add a RapidSpellWebMultiple, and set the RapidSpellWebLaunchers property to the ID of the DataGrid. This will cause RapidSpellWebMultiple to launch all RapidSpellWebLauncher or RapidSpellWInline controls it finds inside the control.

Code Example

```
<form id="DataGrid" method="post" runat="server">
                  <asp:DataGrid id="MyDataGrid" style="Z-INDEX: 101; LEFT: 21px;
POSITION: absolute; TOP: 67px" runat="server" Width="521px" BorderColor="#999999"
BorderStyle="None" BorderWidth="1px" BackColor="White" CellPadding="3"
GridLines="Vertical" AutoGenerateColumns="false"
OnItemCreated="DataGrid_ItemCreated"
                  >
                        <Columns>
                              <asp:TemplateColumn HeaderText="Favourite Films">
                                    <ItemTemplate>
                                          <asp:TextBox id=TextBox1 runat="server"
Text='<%# DataBinder.Eval(Container.DataItem, "Favourite Films") %>'
Height="43px" TextMode="MultiLine" Rows="5" Columns="40" >
                                          </asp:TextBox><BR>
                                          <RapidSpellWeb:RapidSpellWebLauncher
id="RapidSpellWebLauncher" RSMultipleID="RapidSpellWebMultiple1" runat="server"
RapidSpellWebPage="PopUp.aspx" TextComponentName="TextBox1"
ShowButton="false"></RapidSpellWeb:RapidSpellWebLauncher>
                                    </ItemTemplate>
                              </asp:TemplateColumn>
                        </Columns>
                  </asp:DataGrid>

      <rapidspellweb:RapidSpellWebMultiple id="RapidSpellWebMultiple1" style="Z-
INDEX: 105; LEFT: 722px; POSITION: absolute; TOP: 599px"
runat="server" RapidSpellWebLaunchers="MyDataGrid" ShowFinishedMessage="False"></
rapidspellweb:RapidSpellWebMultiple>
            </form>
```

Validators (Spell Checking Before Saving)

It maybe desirable to have the spell checker check the contents of text boxes on the form for errors before proceeding. RapidSpellWebLauncher and RapidSpellWInline can act as Validator controls, in a manner very similar to the standard ASP.NET validators. When they are enabled as validators (by setting the Validator property to true) they will warn the user about any errors and demand that spell checking is at least performed. The user can leave spelling 'errors' in the text box (because some words, such as names, are not known by the spell checker, but are valid), but they must at least run the spell check if there are errors.

Set the AutoLaunchWhenTextInvalid property to true to have the spell check run automatically after the page posts back - note AutoLaunchWhenTextInvalid may not work with RapidSpellWebLauncher if

popup blockers are being used in the visitors browser, since most popup blockers will block any popups that they cannot determine were caused by the user directly - RapidSpellWInline doesn't use popups and doesn't suffer from this.

Code Example

```
<form id="Form1" method="post" runat="server">
      <asp:textbox id="TextBox1" style="Z-INDEX: 100; LEFT: 123px; POSITION:
absolute; TOP: 115px" runat="server" Font-Size="11pt" Font-
Names="Tahoma,Verdana,Arial" Width="624px" Height="270px"
TextMode="MultiLine">This demonstratess how to use RapidSpellWInline as a
validator.</asp:textbox>

      <asp:button id="Button1" style="Z-INDEX: 103; LEFT: 579px; POSITION:
absolute; TOP: 401px" runat="server"
                        Text="Save"></asp:button>

      <asp:validationsummary id="ValidationSummary1" style="Z-INDEX: 104; LEFT:
101px; POSITION: absolute; TOP: 458px" runat="server" Font-Size="9pt" Font-
Names="Arial"></asp:validationsummary>

      <RAPIDSPELLWEB:RAPIDSPELLWINLINE id="RapidSpellWInline1" style="Z-INDEX:
105; LEFT: 637px; POSITION: absolute; TOP: 401px" runat="server"
ButtonTextSpellMode="Edit" AutoLaunchWhenTextInvalid="True"
RapidSpellWInlineHelperPage="../rswihelper.aspx" TextComponentID="TextBox1"
Validator="True"></RAPIDSPELLWEB:RAPIDSPELLWINLINE>


</form>
```

In this example the user is queried to run a spell check if there are spelling errors in the box when the form is submitted.

Non-GUI Spell Checker

The RapidSpellChecker class is of particular use in non-GUI scenarios, however it can also be used with custom spell checker interfaces. Below is a very simple excerpt of how this component can be used, please consult the API documentation for further detail.

Code Example

```
.........
      RapidSpellChecker c = new RapidSpellChecker();
      BadWord badWord;
      ArrayList suggestions;

       //check some text.
      c.Check("This is sume text.");

      //iterate through all bad words in the text.
      while((badWord = c.NextBadWord())!=null){

            Console.WriteLine(badWord.GetWord() +
                  "- is not spelt correctly. Suggestions:");

            try{
                  //get suggestions for the current bad word.
                  suggestions = c.FindSuggestions();

                  //display all suggestions.
                  for(int i=0; i<suggestions.Count; i++){
                        Console.WriteLine(suggestions[i]);
                  }

                  //change the bad word in the text with "replacement".
                  c.ChangeBadWord("replacement");

            } catch (NoCurrentBadWordException e){
                  Console.WriteLine(e);
            }

      }
      Console.WriteLine(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 allows keys for "devServer", "qaServer" and "prodServer" if they are accessed through non local browsers (note; 'devServer' and 'qaServer' keys will be provided free), and since it is not acceptable to change the "LicenseKey" property in source code for each server we suggest using the web.config file. Since each server should have a unique web.config file, storing the license keys in there makes sense.

Store the license key in the web.config in the <appSettings> section (if it doesn't exist, create it after the system.web section).

Eg;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
      <system.web>
            ...........
      </system.web>
      <appSettings>
            <add key="Keyoti-RapidSpellWeb-LicenseKey" value="XYZXYZXYZ"/>
      </appSettings>
</configuration>
```

The key must be named "Keyoti-RapidSpellWeb-LicenseKey" or "Keyoti-RapidSpellWeb-LicenseKey2" if there are two keys for the server.

Now set each server's web.config accordingly, remember web.config files should not be deployed to a server with an application (because they are unique).

Suggestions Algorithm

It is possible to choose the algoritm used to find suggestions for spelling errors, there are two algo-

rithms on offer, 'hashing' (default) and 'phonetic'. Each algorithm has it's own advantages and disadvantages;

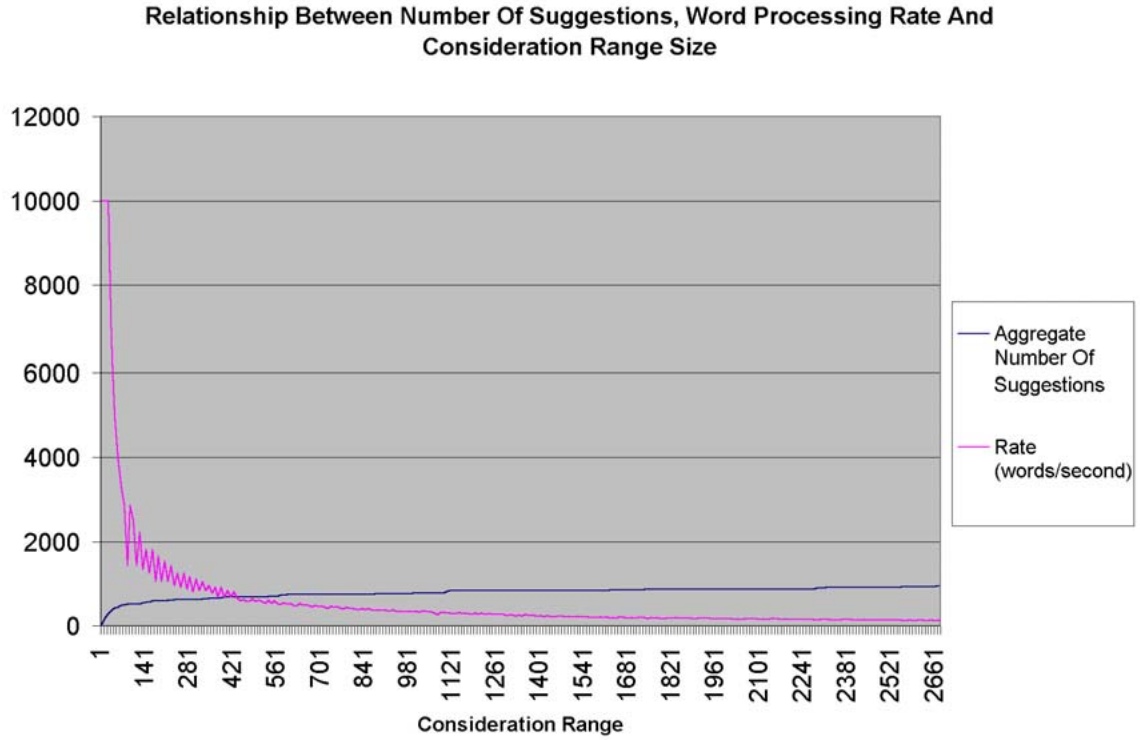
Hashing (default): Very fast, roughly 40 times faster than the phonetic algorithm, works best on typographic errors and common spelling mistakes. Similar to most spell checker suggestions.

Phonetic (Sounds like, English only): Slower, works well with genuine attempts to spell correctly, works poorly with typographics 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 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 & Functions

RapidSpell Web provides an interface to the various events and functions present on the client side in Javascript (ECMAscript). This allows further behaviour customisation when the spell checker is used in popup mode. The current highlights are;

Notification of the spell check finished event
Notification that the user has corrected a word
Exposure of the Javascript function that launches the spell checking

FinishedListener - Finished Event

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 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, closing the window, or by clicking the 'Finish' button). The function must not expect arguments, unless the FinishedListenerInformState is set true and be with in the scope of the whole page (if FinishedListenerInformState==true then the argument is true if spell check completed, false if cancelled).

For example;

```
..............
<script>
      function notifyDone(){
            alert('Finished event captured.');
      }
</script>
..............
<RapidSpellWeb:RapidSpellWebLauncher id="rapidSpellWebLauncher" runat="server"
      TextComponentName="myForm.sourceTextBox"
      TextComponentInterface="Standard"
      FinishedListener="notifyDone"
      Mode="popup"
      SeparateHyphenWords=true
      RapidSpellWebPage="RapidSpellCheckerPopUp.aspx"/>
..............
```

CorrectionNotifyListener - User Correction Event

To be notified every time that the user has corrected a word, set the CorrectionNotifyListener property of RapidSpellWebLauncher to the name part of a Javascript function present on the main page.

For example;

```
..............
<script>
      function OnCorrection(startIndex, oldWord, newWord){
            alert('Correction occured to word at index '+startIndex+' old word
was '+oldWord+' replacement text is '+newWord);
      }
</script>
..............
<RapidSpellWeb:RapidSpellWebLauncher id="rapidSpellWebLauncher" runat="server"
      TextComponentName="myForm.sourceTextBox"
      TextComponentInterface="Standard"
      CorrectionNotifyListener="OnCorrection"
      Mode="popup"
      RapidSpellWebPage="RapidSpellCheckerPopUp.aspx"/>
..............
```

Function To Launch Checking In RapidSpellWebLauncher

The popup spell checking is started by a Javascript function whose name is dependent on the ID of the RapidSpellWebLauncher. When the function is called, the spell checking is started just as if the button was clicked, all the control properties from design time are used. The name and pattern of the function is;

```
popUpCheckSpellingXXXXX('rsTCIntXXXXX')
```

where XXXXX is the ID of the RapidSpellWebLauncher control.

Function To Launch Checking In RapidSpellWInline

To start spell checking from Javascript in the RapidSpellWInline use the following pattern.

```
XXXXX_SpellChecker.OnSpellButtonClicked()
```

where XXXXX is the ID of the RapidSpellWInline control.

Switching Language

It is possible to set the text used in the GUIs to one of several preset languages. Setting the GuiLanguage property in RapidSpellWebLauncher/RapidSpellWInline will change the text of the button and also pass on that value automatically to RapidSpellWeb. To customize button or label text, set the button value or label text properties accordingly, this will override the GuiLanguage set text. In VisualStudio, if the property has been changed and you want to reset the text to default, simply delete the entire field, this will reset the text to default. Due to RapidSpellWeb receiving it's GuiLanguage setting from RapidSpellWebLauncher, it is not possible for changes to GuiLanguage to be reflected in the RapidSpellWeb Control's appearance at design-time, this is a design-time only limitation, at run time the text will be in the chosen language.

To switch the UI and dictionary language set the RapidSpellWebLauncher/RapidSpellWInline properties GuiLanguage, LanguageParser and DictFile in either code-behind or a script block on the form.

Code Example Language Switcher

Using a select box to select the UI language and dictionary.

```
<%@ Register TagPrefix="rapidspellweb" Namespace="Keyoti.RapidSpell"
Assembly="Keyoti.RapidSpellWeb" %>
<%@ Page language="c#" AutoEventWireup="true" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
      <script language="C#" runat="server">
      void LanguageChanged(object sender, EventArgs e){
            switch (LanguageList.SelectedItem.Value){
                  case "English":
                        RapidSpellWebLauncher1.GuiLanguage =
Keyoti.RapidSpell.LanguageType.ENGLISH;
                        RapidSpellWebLauncher1.LanguageParser =
Keyoti.RapidSpell.LanguageType.ENGLISH;
                        //Set dictionary file path on server here
                        RapidSpellWebLauncher1.DictFile = "path\to\english.dict";
                  break;
                  case "Español":
                        RapidSpellWebLauncher1.GuiLanguage =
Keyoti.RapidSpell.LanguageType.SPANISH;
                        RapidSpellWebLauncher1.LanguageParser =
Keyoti.RapidSpell.LanguageType.SPANISH;
                        //Set dictionary file path on server here
                        RapidSpellWebLauncher1.DictFile = "path\to\spanish.dict";
```

```
                  break;
                  case "Français":
                        RapidSpellWebLauncher1.GuiLanguage =
Keyoti.RapidSpell.LanguageType.FRENCH;
                        RapidSpellWebLauncher1.LanguageParser =
Keyoti.RapidSpell.LanguageType.FRENCH;
                        //Set dictionary file path on server here
                        RapidSpellWebLauncher1.DictFile = "path\to\french.dict";
                  break;
            }
      }
      </script>
      <body>
            <form id="Form1" method="post" runat="server">
                  <asp:textbox id="TextBox1" runat="server" Width="288px"
TextMode="MultiLine" Height="216px"></asp:textbox>
                  <rapidspellweb:rapidspellweblauncher
                        id="RapidSpellWebLauncher1"
                        runat="server"
                        LookIntoHyphenatedText="True"
                        LanguageParser="ENGLISH"
                        RapidSpellWebPage="RapidSpellCheckerPopUp.aspx"
                        TextComponentName="TextBox1" >
                  </rapidspellweb:rapidspellweblauncher>
                  <asp:DropDownList OnSelectedIndexChanged="LanguageChanged"
AutoPostBack="True" id="LanguageList" runat="server" Width="178px">
                        <asp:ListItem Text="English" Value="English" selected />
                        <asp:ListItem Text="Español" Value="Español" />
                        <asp:ListItem Text="Français" Value="Français" />
                  </asp:DropDownList>
            </form>
      </body>
</HTML>
```

TextComponentName Property Details

The TextComponentName property in RapidSpellWebLauncher is designed to accept identifying strings to identify text controls (or html elements) in a multitude of ways (it automatically detects the method used), it is important to understand how it works if you plan to set this property beyond simply the Control ID of the text box.

There are 3 ways to set this property;

1. Control ID - the simplest way is to set this property to the *server-side* ID of the control to be spell checked. This comes with a caveat; because server-side Control IDs need only be unique with-in a 'naming container' (eg. page, DataList, DataGrid) and because the Control must be found via the FindControl method (which searches the naming container that the RapidSpellWebLauncher control is in) **it is only possible to use the Control ID to specify the text box if the text box Control and the RapidSpellWebLauncher are in the same naming container.**

2. Control ClientID - setting the TextComponentName to the ClientID of the text control is a good way to firmly identify the text Control to be spell checked. To do this it is usually best to set the TextComponentName property in code-behind or with a data-bind using the ClientID of the text Control.

eg.

```
rapidSpellWebLauncher.TextComponentName = textBox1.ClientID
```

This method can also be used when referring to traditional Html elements (such as <textarea> or <input type=text>), in this case the ID attribute of the tag is used to identify the element.

eg.

```
<textarea ID="textbox1">This is my text box</textarea>
.....
rapidSpellWebLauncher.TextComponentName = "textbox1"
```

The only warning with this method is to remember (as per good asp.net practice) that the ClientID property of a control is not properly defined until run-time (when the page loads in the browser), therefore the TextComponentName property should be set at run-time, also.

3. Javascript formName.textBoxName - traditionally in client side coding the <formName>.<textBoxName> style was used to identify form elements.

eg.

```
<form name="mainForm">
      <textarea name="textbox1"></textarea>
</form>
.....
TextComponentName = "mainForm.textbox1"
```

This method can still be used with RapidSpell Web, however in more complex cases (unusual Control IDs, DataGrids etc) it can be hardest to use.

Note: if the "Custom" Javascript interface is being used for a text box interface then the tbElementName argument in

```
RSCustomInterface(tbElementName)
```

will be set according to the method used to identify the text box, as detailed below for each method;

1. Control ID - tbElementName will be set to the ClientID of the Control given by TextComponentName.

2. Control ClientID - tbElementName will be set to TextComponentName.
3. Javascript formName.textBoxName - tbElementName will be set to TextComponentName.

Performance Tips

Performance can be maximized very readily;

Dictionary Caching

If the application uses only one main dictionary (i.e. there are no Dict Files being switched at runtime) then enabling caching will improve performance by roughly 50 to 100% (the exact amount depends on the amount of text being checked), as this will remove the need to reload the dictionary for each request. To do this set CacheDictionary to true in the RapidSpellWeb and RapidSpellWInlineHelper controls.

Parser

If V2Parser is set true in RapidSpellWebLauncher/RapidSpellWInline then there is an automatic 10-20% performance hit, this is because although the V2Parser is more accurate, it is also slightly slower. We recommend the use of the V2Parser, but if performance is an issue this can be set to false. Note that V2Parser=true is needed for URL/Email ignoring.

Dict Files

Dict Files do load quicker than DLL dictionaries, therefore they are worth using if Dictionary Caching is disabled.

CAS (Code Access Security)

The simplest way to use RapidSpell Web DLLs in a non Full trust environment is to add the DLLs to the GAC (by default GAC assemblies are given full trust). Otherwise the following permissions are required;

```
      <IPermission
      class="AspNetHostingPermission"
      version="1"
      Level="Low"
      />

      <IPermission
      class="FileIOPermission"
      version="1"
      Unrestricted="true"
      />

      <IPermission
      class="SecurityPermission"
      version="1"
      Flags="Assertion, Execution, ControlThread, ControlPrincipal,
SerializationFormatter"
      />
```

Note this is roughly equivalent to High or Medium trust with the addition of the **SerializationFormatter** attribute for SecurityPermission (this is required to read the dictionary from the Keyoti.RapidSpellMDict.DLL). Write FileIOPermission is only required if using user dictionaries, and can be restricted to the directory where the file is written, read permission is required if Dict File dictionaries are being used.

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!

Troubleshooting

If you do run in to trouble, please consult the following for some tried and tested solutions to issues that have occured with RapidSpell Web .NET - also our KB is online at http://keyoti.com/kb.

Problem	Solution
User dictionary; the 'Add' button changes to 'Adding...' when it is clicked and nothing else happens.	This is an indicator that an exception occured during the control's attempt to write to the user dictionary. Double check that the ASP_NET process has permission to write to the file and folder specified for the user dictionary file. If unsure, try changing the path. Remember, just setting the user dictionary path to a relative path (eg. "userdict.txt") will result in a path off the Windows system folder, not the current web app folder, use HttpRequest.MapPath to map a virtual path.
Visual Studio fails to build the project sometimes and reports errors such as **BC31011: Unable to load referenced library.**	Although theoretically it is not required, the simplest solution to this problem is adding the RapidSpell DLLs to the GAC on the development machine. This is a workaround to an issue with the development environment, not RapidSpell. To do this use GACUTIL (comes with the .NET framework) in the same folder as your DLLs: gacutil /i Keyoti.RapidSpellWeb.dll gacutil /i Keyoti.RapidSpellMDict.dll and then re-add the references to the project.
A license error is shown.	Ensure that the LicenseKey property in RapidSpellWeb (the control in the popup, usually) is set. Ensure that the key has no white space or newlines in it. Ensure that the key is a 'deployment key' generated through our website, not an 'activation key'. Ensure that the page on the server is up to date and has the key set (this is often wrongly assumed).

If you are having other problems, you are free to participate in our support forum, or email us at support@keyoti.com