Spell-Checker for ASP.NET using Word

**Summary:** This article explains how to create a spell-checker for ASP.NET using Microsoft Office Word's spell-checking capabilities. (7 printed pages)

*Source code is provided in C# and VB for FREE.*

**Overview**

Microsoft Word includes a lot of resources such as a dictionary, a thesaurus, and a spell-checker. You can use these tools for spelling and grammatical changes in order to create accurate and credible documents.

While you can automatically use the spell-checker inside Word, you can also use its object model to utilize the same functionality in your own applications. This article shows how to use these objects and methods to implement a custom spell-checker for ASP.NET using the Microsoft C# programming language.

**Spell-Checking Capabilities of Word**

Word provides an object model and a set of methods to help you spell-check a piece of text programmatically in both Windows and Web applications. You can actually create a spell-checking engine that can be commonly used by both types of .NET applications.

First you need to create a new Word application object using COM Interoperability as in the following code segment:

```csharp
Word.Application app = new Word.Application();
Word._Document doc = app.Documents.Add(ref template, ref newTemplate, ref documentType, ref visible);
```

Word includes two main objects for spell-checking: **ProofreadingErrors** and **SpellingSuggestions**. The ProofreadingErrors collection is a property of the Range object and contains the misspelled words in the specified text:

```csharp
doc.Words.First.InsertBefore(textToSpellCheck);
Word.ProofreadingErrors spellErrorsColl = doc.SpellingErrors;
```

Now you can read the misspelled words from the ProofreadingErrors collection, and get a list of suggestions for each word into a **SpellingSuggestions** object by calling the **GetSpellingSuggestions** method as in the following code segment:
Spell-Checker Using Word

It’s pretty straightforward to use the spell-checking functionality of Word in a Windows .NET application. All you have to do is to create a Windows Form with several TextBox, ListBox and CommandButton controls on it, and to use the Word objects and method calls as explained in the previous section.

However, it’s a little bit tricky to apply the same approach to an ASP.NET web application due to its stateless nature. You need to create a web form with similar TextBox, ListBox and Button server controls on it, but you have to pay special attention to the interaction of this web form with its parent window that contains the text to be spell-checked. Figure 1 illustrates the user interface provided by the parent window web form.

![Figure 1. Parent Window Web Form](image)

When the user clicks the Spell Check button it will pop up the spell-checker web form, and pass the ID of the TextBox control to be spell-checked as in the following code segment:

```javascript
window.open("SpellCheck.aspx?ctl=TextBox1", "SpellCheck",
'height=230,width=280,location=no,menubar=no,resizable=no,scrollbars=no,
status=yes,toolbar=no');
```
Spell-checker web form gets the value of the TextBox control in the parent window, keeps it in a HiddenField, and performs a spell-check on the whole text using Word’s object model. Figure 2 illustrates the user interface provided by the spell-checker web form.

![Figure 2. Spell-Checker Web Form](image)

After creating the Word application and document objects, it sets the language to **English**. You can change the following statement to spell-check in any language supported by Word:

```csharp
```

As soon as it completes the spell-checking process, it saves all the spelling errors with their suggestions, number of errors, current error index and current error position in **Session** variables as in the following code segment:

```csharp
Session.Add("Errors", strErrors);
Session.Add("ErrorCount", errorCount);
Session.Add("ErrorIndex", 1);
Session.Add("ErrorPosition", 0);
```

Then it opens the spell-check web form, and displays the first error along with its suggestions in the ListBox below. The first suggestion is automatically written into the TextBox. When the user selects a different suggestion in the ListBox, it is written
into the TextBox so that the user can easily modify it if necessary. This feature is achieved with a combination of server-side and client-side coding:

**Server-Side**

```csharp
lbSuggestions.Attributes.Add("onchange", "Suggestions_OnChange(this)");
```

**Client-Side**

```javascript
function Suggestions_OnChange(lbElem) {
    document.getElementById('txtChangeTo').value = lbElem.value;
}
```

It displays one misspelled word at a time, and lets the user fix that error. Ignore button gets the user to the next error, while the Change button changes that particular error with the specified alternative in the TextBox.

If the user clicks the Ignore All button it will add the misspelled word to an ArrayList to ignore all occurrences of the same error. The ArrayList object will be saved in a Session variable so that it doesn’t get lost during page refreshes:

```csharp
alIgnoreAll = (ArrayList)Session["IgnoreAll"]; if (!alIgnoreAll.Contains(lblMisspelledWord.Text)) {
    alIgnoreAll.Add(lblMisspelledWord.Text);
    Session["IgnoreAll"] = alIgnoreAll;
}
```

If the user clicks the Change All button it will add the misspelled word and the value in the TextBox in a Hashtable to replace all occurrences of the same error with the same alternative. The Hashtable object will be saved in a Session variable.

```csharp
htChangeAll = (Hashtable)Session["ChangeAll"]; if (!htChangeAll.Contains(lblMisspelledWord.Text)) {
    htChangeAll.Add(lblMisspelledWord.Text, txtChangeTo.Text);
    Session["ChangeAll"] = htChangeAll;
}
```

Note that the Change All button doesn’t immediately update the text, but waits until the user clicks the OK button. When the user clicks the OK button, it performs a final
spell-check operation behind the scenes and updates all words in the ChangeAll Hashtable. You can introduce a different logic to avoid a second spell-check operation at the end, but you have to be careful about keeping track of the character position shifts between the original text and the new text properly.

User may click the OK or Cancel button at any time. However, if the user finishes processing all the spelling errors it will display a “Spell check complete” message, and disable all the buttons other than OK and Cancel.

Cancel button will close the spell-check window without making any changes to the parent window. OK button will update the TextBox on the parent window web form with the new text, and then close itself.

Limitations

Even though Spell-Checker Using Word will help you fulfill your basic spell-checking needs in your ASP.NET applications, you should take a look at the following limitations before considering a production level application development.

**Microsoft Office Required**: You need to install Microsoft Office on the server in order to use the spell-checking functionality through its object model. Therefore you need to acquire a license for Microsoft Office on your server.

**Additional Coding Required**: You need to write some code in your web pages in order to open up the spell-check web form and pass in the identifier of the control that contains the text to spell-check.

**COM Interoperability**: You need to create the Word application object using COM Interoperability because there is no native .NET library provided for Word. This may become an issue for garbage collection of those objects.

**Scalability**: You need to load the Word document object every time a new spell-checking session is initiated. This may not scale well when you have a lot of users connecting to website and running the spell-checker.

**User Experience**: It displays only the misspelled word without the context. This makes it hard to figure out where the error is in the actual text. You should provide a better user interface to show the misspelled word in the sentence or paragraph.

**Page Refresh**: Every time you click one of the buttons on the spell-check web form, it refreshes the whole page because of the postback to server. This may not be an issue for a short text, but it could be pretty annoying for a long text.

**Custom Dictionary**: Sometimes the user wants to add certain words into a custom dictionary because they are not actually misspelled words. You may need to add a new button on the form to let the user add a word to a custom dictionary at runtime.
Spell-check HTML: Even though you can spell-check regular text, you need to write your own code to spell-check HTML. Otherwise it spell-checks the words inside HTML tags as if they are part of the text.

Ignore Special Words: You may need to add new features to ignore e-mail and Internet addresses, words in uppercase, words with numbers, and so on in order to avoid unnecessary spelling-errors.

Repeated Words: Word catches repeated words as spelling-errors, and suggests an empty string to replace it with. However, this leaves an extra space before or after the repeated word that gets deleted.

Change All: This requires an additional spell-check operation when the user clicks the OK button at the end. You may consider an alternative solution to eliminate this extra step, but keep the caution on character position changes when you replace all words.

Karamasoft UltimateSpell

If you need a more elaborate solution we strongly recommend you visit Karamasoft UltimateSpell at http://www.karamasoft.com/?component=UltimateSpell.

Here is a comparison of Karamasoft UltimateSpell with Spell-Checker Using Word:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Karamasoft UltimateSpell</th>
<th>Spell-Checker Using Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display spelling errors and suggestions</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Provide Ignore and Change functionality</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Display OK and Cancel buttons</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Use spelling engine in Web and Windows</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Multilingual spell-check</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Work without a Word license on server</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Work without any coding on web pages</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Work without COM Interoperability</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Work as an ASP.NET server control</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>AJAX-enabled to avoid page refresh</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Spell-check as you type</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Spell-check HTML</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
Spell-check selected area | ✓ | ✗ |
| Auto correct spelling errors | ✓ | ✗ |
| Auto find editable controls | ✓ | ✗ |
| Scalable for long text and heavy traffic | ✓ | ✗ |
| Cache dictionary in memory | ✓ | ✗ |
| Display misspelled word in sentence | ✓ | ✗ |
| Ignore special and compound words | ✓ | ✗ |
| Add words to custom dictionaries | ✓ | ✗ |
| Look up meaning of suggestions | ✓ | ✗ |
| Server-side API for event handling | ✓ | ✗ |
| Client-side API for spell-checking | ✓ | ✗ |
| Work inside User Control and DataGrid | ✓ | ✗ |
| Multilingual dialog box | ✓ | ✗ |
| Integrated into UltimateEditor & UltimateSearch | ✓ | ✗ |

**Conclusion**

Whether you are building Windows or Web applications using the Microsoft .NET Framework, you can easily use the objects and methods of a specific Microsoft Office program to add powerful features with minimum coding.

However, you should be aware of the limitations of these tools for a professional-grade application to be deployed onto production servers. You have to pay special attention to licensing, performance and usability issues before initiating your projects.

If you are looking for a basic spell-checker in your entry-level projects Spell-Checker Using Word will fulfill your needs without any doubt. As soon as your needs exceed its limitations you can always rely on Karamasoft UltimateSpell in order to develop quality Windows and Web applications in .NET.