ComponentOne

SpellChecker for WinForms

ComponentOne, a division of GrapeCity
201 South Highland Avenue, Third Floor
Pittsburgh, PA 15206 USA

Website: http://www.componentone.com
Sales: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Table of Contents

SpellChecker for WinForms Overview

SpellChecker for WinForms, our .NET product for multi-language spell checking, has been re-engineered to provide the easiest-to-use and most efficient spell checker available on the market today. Just drop the **C1SpellChecker** and text box controls on your Windows Form, set one property, and experience Microsoft Word-like spell checking capable of processing 400,000 words per second!

C1SpellChecker adapts to your organization's needs – providing as-you-type spell checking, multi-language support, full customization, and more. Take advantage of the new functionality, speed, and flexibility – start using **C1SpellChecker** today.

Help with WinForms Edition

Getting Started

For information on installing **ComponentOne Studio WinForms Edition**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with WinForms Edition.

Key Features

The **C1SpellChecker** replaces the **C1Spell** component offering substantial new features, including:

- **Code-free text box integration**
  Use the extender **SpellCheck** property to link the **C1SpellChecker** component to any text box-derived control – no code is necessary!
- **Fastest SpellChecker**
  Fastest **SpellChecker** capable of processing 400,000 words per second.
- **As-you-type spell checking**
  As you type, a red, wavy underline indicates any spelling mistakes; right-clicking the error shows a context menu with spelling suggestions.
- **Dialog box spell checking**
  Use the **CheckControl** method to check the content of any text box using a modal dialog box – C1SpellChecker automatically highlights suspect words.
- **Spell checking intelligence**
  Differentiate between upper and lower case; therefore, "paul" is marked as a spelling mistake, and "Paul" is not.
- **Full dialog box customization**
  Incorporate your organization's look and feel – add customizable images, text, buttons, and more to the default spelling dialog box.
- **Interactive context menu**
  Right-click a misspelled word and the default context menu offers a list of suggestions, the option to Ignore All, use AutoCorrect, or choose a word from the spelling dialog box.
- **Spell checking support for other types of controls**
  Supports programmatic interfaces that can be used to add spell checking functionality to other kinds of controls, such as grids.
- **Flexibility to create custom dictionaries**
  Using the dictionary editor (C1DictionaryEditor.exe), create and maintain custom-built dictionary files to distribute with your application – helpful for words specific to certain industries.
- **Multi-language support**
  Includes 16 international dictionaries.
- **Spell check strings of code**

  Create custom text parsers to spell check C# and VB code files, for example.

SpellChecker for WinForms Quick Start

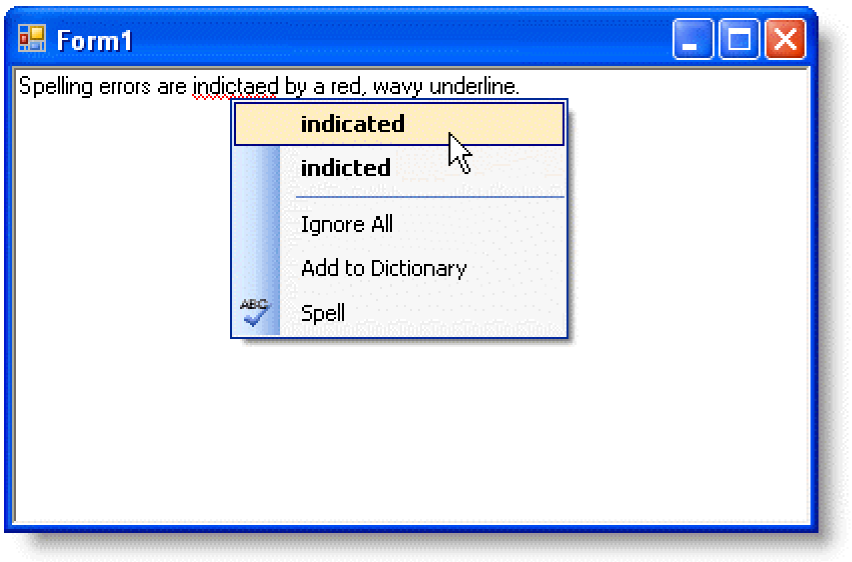
This section will lead you through the creation of a basic Visual Studio project that uses the **C1SpellChecker** component with a RichTextBox control. Note that you could also check the spelling of the text in any TextBox control.

Complete the following steps to create a basic spell-checking form:

1. Add the **C1SpellChecker** component to the Toolbox.
2. From the Toolbox, double-click the **C1SpellChecker** component to add it to the form. Note that the component will appear below the form, not on it.
3. From the Toolbox, double-click the RichTextBox control to add it to the form.
4. Select the RichTextBox, and set the following properties:
   - **Dock** property to **Fill**.
   - **SpellChecking on C1SpellChecker1** property to **True**.

Note: The **SpellChecking** property is an extender property provided by the **C1SpellChecker**.

Now the sample application is ready; it’s that easy! Run the application, type any text in the box, and observe Microsoft Word-style spell-checking functionality (right-click misspelled words to see a list of suggestions):



Spelling errors are indictaed by a red, wavy underline.

Notice that spelling errors are indicated by a red, wavy underline. Right-clicking the spelling error provides valid spelling alternatives (just like in Microsoft Word).

C1SpellChecker Fundamentals

This section covers the essentials of **C1SpellChecker**. After reading the following topics you will have a good understanding of how the **C1SpellChecker** operates.

Modes of Spell-Checking

The **C1SpellChecker** component supports the following three modes of spell-checking:

- **Batch mode**
  Use the CheckText, CheckWord, and GetSuggestions methods to check strings and get lists of errors and spelling suggestions.
- **Dialog mode**
  Use the CheckControl method to check the content of any Windows.Forms.TextBoxBase-derived controls using a modal dialog box. The **C1SpellChecker** will check the text and display a dialog box where the user can choose to correct or ignore each error. See the Built-in Spell Dialog Box topic for details.
  You can also check editors that do not derive from Windows.Forms.TextBoxBase. To do that, you have to create a wrapper class that implements the ISpellCheckableEditor interface.
- **As-you-type mode** Set the extender property **SpellCheck** on any Windows.Forms.TextBoxBase-derived control to **True**, and the **C1SpellChecker** will monitor changes to the control. Any spelling mistakes will be indicated on the control by a red, wavy underline; right-clicking the errors will show a context menu with spelling suggestions.
  You can also provide as-you-type spelling support for editors that do not derive from Windows.Forms.TextBoxBase. To do that, you have to create a wrapper class that implements the ISpellCheckableRichEditor interface.

In all three modes described above, the spell-checker follows rules that can be customized using the Options property. Options available include types of words to ignore (capitalization, numbers, URLs), whether to display suggestions in a context menu, the number of suggestions to display, and so on.

Types of Dictionary Files

The **C1SpellChecker** control uses up to three dictionaries while checking text:

- **Main dictionaries:** Read-only dictionary that contains the main word list. The US-English version of this dictionary is built into the control, so there's no need for any additional files. Other languages are available as .dct files that ship with the control, and can be selected using the FileName property.
- **User dictionaries:** Read-write dictionary used to store words that are correct, but are not part of the main dictionaries. These files are stored as plain UTF-8 text, and can be selected using the FileName property.
- **Custom dictionaries:** Any .NET object that implements the ISpellDictionary interface. This allows users to create their own dictionary classes, using whatever scheme makes sense in their application. A custom dictionary could, for example, look up words on the Web using a Web service (and then cache them for speed).

Word Lists

The main dictionaries are zip files with a .dct extension. The zip file may contain several word lists, each one stored as a UTF-8-encoded text file containing lists of valid words. All such entries must have a ".words" extension. For information on how to add word lists, see the Editing the Contents of the DCT File and Creating a New DCT File topics.

Rules

The file may also include a "rules" entry that specifies rules to apply when spell-checking text in the dictionary language. For example, the French dictionary that ships with **C1SpellChecker** contains the following entries:

- IgnorePrefix: l' d' j' da' m' s' n' qu'
- IgnoreSuffix: 's

These tell the spell checker to ignore some common prefixes and suffixes; they are removed before the word is checked. For example:

- · l'amour (check 'amour' -> correct)
- · l'amuor (check 'amuor' -> incorrect)
- · Maxim's (check 'Maxim' -> correct)
- · Naxim's (check 'Naxim' -> incorrect)

Prefixes and suffixes not included will be tagged as spelling errors:

- h'amour (check 'h'amour' -> incorrect)
- x'amuor (check 'x'amuor' -> incorrect)

Spell-Checking International Applications

C1SpellChecker has a built-in American English dictionary. To spell-check text in other languages, use the MainDictionary property and set the FileName property to the name of the dictionary you want to use.

ComponentOne ships the following 15 international dictionaries with **C1SpellChecker**, in addition to the English – US dictionary that is built into the control:

C1Spell_de-DE.dct	German dictionary.
C1Spell_el-GR.dct	Greek dictionary.
C1Spell_en-AU.dct	English – Australia dictionary.
C1Spell_en-CA.dct	English – Canada dictionary.
C1Spell_en-GB.dct	English – Great Britain dictionary.
C1Spell_en-US.dct	English – US dictionary, built into the control as well.
C1Spell_es-AR.dct	Spanish – Argentina dictionary.
C1Spell_es-ES.dct	Spanish – Spain dictionary.
C1Spell_es-MX.dct	Spanish – Mexico dictionary.
C1Spell_fr-CA.dct	French – Canada dictionary.
C1Spell_fr-FR.dct	French – France dictionary.
C1Spell_it-IT.dct	Italian dictionary.
C1Spell_nl-NL.dct	Dutch dictionary.
C1Spell_pt-BR.dct	Portuguese – Brazil dictionary.
C1Spell_pt-PT.dct	Portuguese – Portugal dictionary.
C1Spell_ru-RU.dct	Russian dictionary.

Note: By default, the DCT files are installed in the **C:\Program Files\ComponentOne\WinForms Edition\bin** directory.

If your application uses any dictionaries other than the built-in American English, then you need to deploy the dictionaries with the application.

Deploying Dictionaries

Deploying dictionaries is trivial for English applications since the English dictionary is built into the **C1SpellChecker** component. Other languages are available, but require deploying the appropriate dictionaries.

The easiest way to deploy the dictionaries with your application is to add the .dct files to your project, and set the **Build Action** property to **None** and the **Copy to Output Directory** property to **Copy if newer**. This will place the .dct files in the application directory where **C1SpellChecker** can find them.

When using this deployment method, make sure the main dictionary's FileName value specifies a file name without a path. This way, the component will search for the dictionary in the directory where the **C1SpellChecker** assembly is located.

By default, **C1SpellChecker** will also localize the built-in spell dialog box automatically, based on the current culture. You can override this behavior and specify the language used in the dialog box by setting the DialogLanguage property.

Using the C1SpellChecker Dictionary Editor

You can use the dictionary maintenance utility that ships with **C1SpellChecker** to modify the dictionaries that ship with **C1SpellChecker** and also to create new dictionaries. For more details on maintaining and creating dictionaries, see the Building and Maintaining Dictionary Files section.

Spell-Checking Different Types of Controls

C1SpellChecker can spell-check controls that derive from Windows.Forms.TextBoxBase. This includes the TextBox and RichTextBox controls.

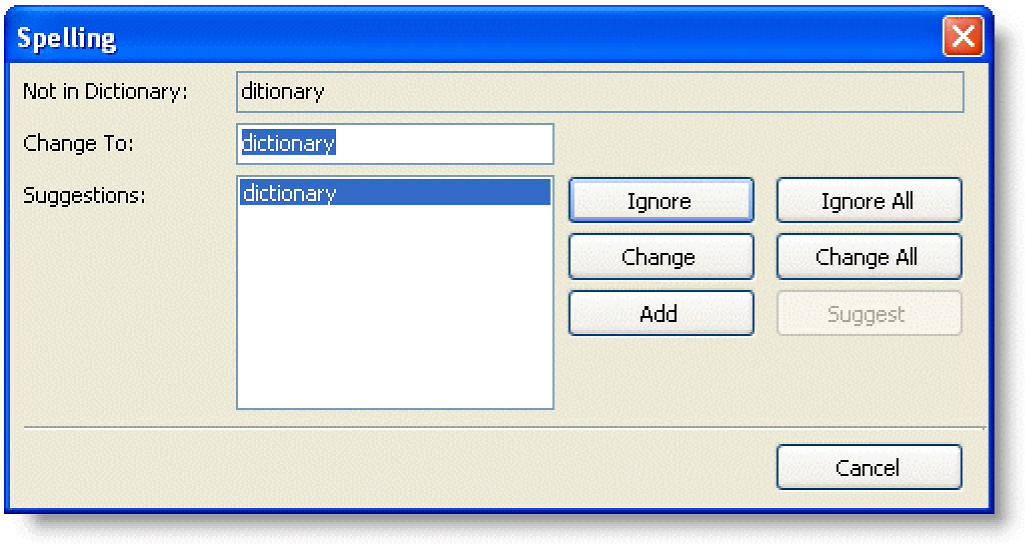
To spell-check other types of controls (a grid for example), you have to create a wrapper class that implements the ISpellCheckableEditor interface or the ISpellCheckableRichEditor interface.

The ISpellCheckableEditor interface specifies the minimum set of methods and properties necessary to implement modal (dialog-base) spell-checking. The ISpellCheckableRichEditor interface extends ISpellCheckableEditor and specifies additional methods needed to provide as-you-type spell-checking (with the red wavy underlines and spelling suggestions in the context-sensitive menu).

Note: The samples that ship with the **C1SpellChecker** control include an application called **SpellGrid** that shows how you can implement the ISpellCheckableEditor interface in a class and use that class to spell-check a **C1FlexGrid** control.

Built-in Spell Dialog Box

Here is the default built-in **Spelling** dialog box:



To enable the **Spell Dialog**, use the CheckControl method to link the **C1SpellChecker** component to a control derived from Windows.Forms.TextBoxBase. For example, the following code links the **C1SpellChecker** component to a RichTextBox control:

To write code in Visual Basic

Visual Basic

```
Private Sub btnSpellCheck_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnSpellCheck.Click
    C1SpellChecker1.CheckControl(RichTextBox1)
End Sub
```

To write code in C#

C#

```
private void btnSpellCheck_Click(object sender, EventArgs e)
{
    c1SpellChecker1.CheckControl(richTextBox1);
}
```

While spell checking, the **C1SpellChecker** will automatically highlight misspelled words when the user interacts with the built-in **Spell Dialog**.

Customizing the Spell Dialog

The **Spell Dialog** can be customized three different ways:

1. Create an instance of the C1SpellDialog class, attach event handlers, then pass the instance of the dialog to the CheckControl method. For example:

   To write code in Visual Basic

   Visual Basic

   ```
   Private Sub btnSpell_Click(ByVal sender As System.Object, ByVal e As
   System.EventArgs) Handles btnSpell.Click
   ```

```
    ' create a spell-checking dialog box
    Using dlg As New C1SpellDialog()
        ' connect event handler
        AddHandler dlg.ErrorDisplayed, AddressOf dlg_ErrorDisplayed
                ' spell-check the RichTextBox control
        C1SpellChecker1.CheckControl(Me.RichTextBox, False, dlg)
    End Using
End Sub
Private Sub dlg_ErrorDisplayed(ByVal sender As Object, ByVal e As EventArgs)
    ' get the C1SpellDialog that fired the event
    Dim dlg As C1SpellDialog = TryCast(sender, C1SpellDialog)
        ' show information about the error currently displayed
    ToolStripStatusLabel1.Text = String.Format("Error {0} of {1}: '{2}'",
dlg.ErrorIndex + 1, dlg.ErrorCount, dlg.CurrentError.Text)
End Sub
```

To write code in C#

C#

```
private void btnSpell_Click(object sender, EventArgs e)
{
     // create a spell-checking dialog box
     using (C1SpellDialog dlg = new C1SpellDialog())
     {
          // connect event
           dlg.ErrorDisplayed += new EventHandler(dlg_ErrorDisplayed);
           // spell-check the RichTextBox
          c1SpellChecker1.CheckControl(this.richTextBox, false, dlg);
     }
}
void dlg_ErrorDisplayed(object sender, EventArgs e)
{
     // get the C1SpellDialog that fired the event
     C1SpellDialog dlg = sender as C1SpellDialog;
      // show information about the error currently displayed
     toolStripStatusLabel1.Text = string.Format("Error {0} of {1}: '{2}'",
     dlg.ErrorIndex + 1, dlg.ErrorCount, dlg.CurrentError.Text);
}
```

Note that the code above assumes that you have added a **Button** control and a **StatusStrip** control with a **ToolStripStatusLabel** to your form at design time.

OR

2. Create a new spell-checking dialog class that implements the ISpellDialog interface. Then pass an instance of the new dialog to the CheckControl method.

   OR

3. If you need more extensive customization, you can create your own spell dialog box and use that instead of the built-in one.

Note: The samples that ship with the **C1SpellChecker** include a **CustomSpellDialog** application that includes

> the source code for two dialogs that you can use as a base for creating your own spell dialog boxes.

Other Spell-Checking Services

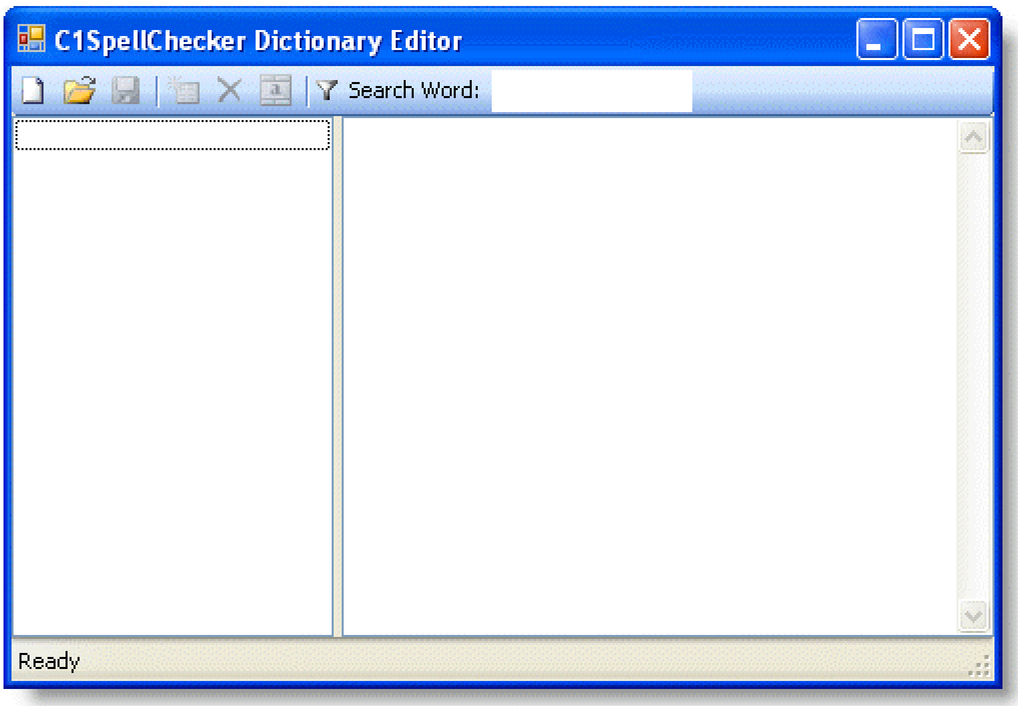
C1SpellChecker exposes lower-level methods for checking text that is not necessarily in any controls. For example, you can use these methods to spell-check text stored in databases or in text files.

The CheckText method spell-checks a string and returns a list of errors in a CharRange object. The GetSuggestions method provides a list of suggestions for misspelled words.

Building and Maintaining Dictionary Files

The **SpellChecker for .WinForms** distribution CD includes a **C1SpellChecker Dictionary Editor** called **C1DictionaryEditor.exe** that allows you to create and maintain dictionary files (.dct). You can use it to add words to the main dictionary or to create new dictionaries in languages other than English.

The installation utility copies the **C1DictionaryEditor.exe** program to the ComponentOne directory you specify during the installation process (by default it installs to the **C:\Program Files\ComponentOne\WinForms Edition\bin** directory). When you run **C1DictionaryEditor.exe**, you will see the following on your screen:



The utility is simple and easy to use. It is used to perform two functions:

- Open existing .dct files and add words or word lists.
- Create new .dct files.

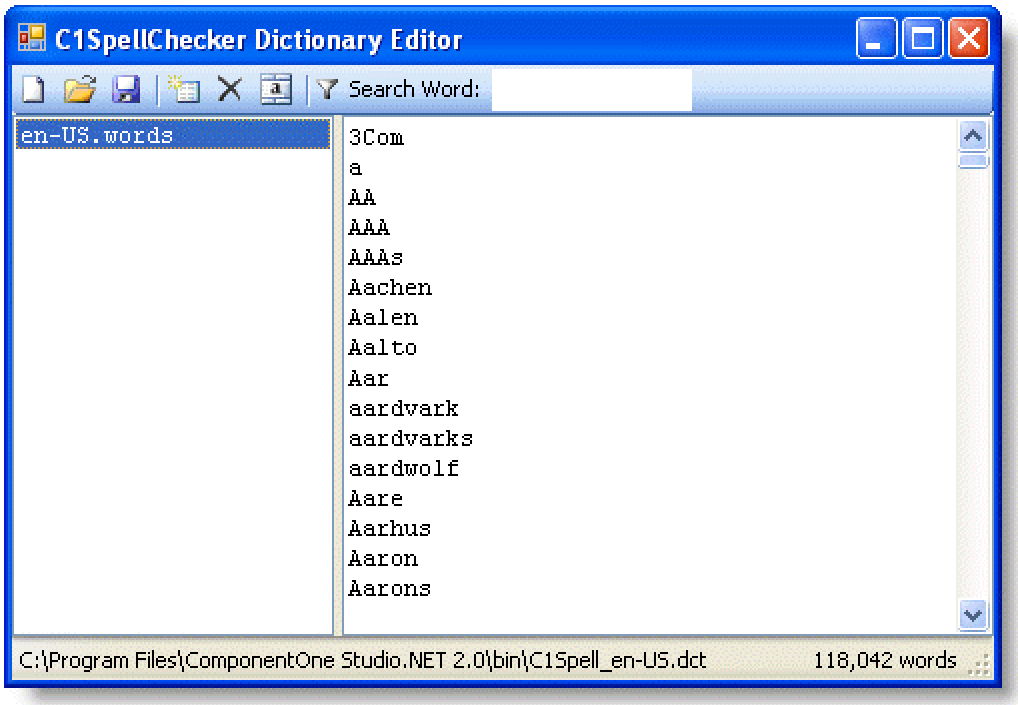
Editing the Contents of the DCT File

To edit the contents of an existing dictionary file (.dct), complete the following steps:

1. Double-click the **C1DictionaryEditor.exe** (by default it installs to the **C:\Program Files\ComponentOne\WinForms Edition\C1SpellChecker**) to open the **C1SpellChecker Dictionary Editor**.
2. Click the **Open Dictionary** button and browse for the .dct file to edit.

   Note: For a list of .dct files that ship with **C1SpellChecker**, see the Spell-Checking International Applications topic.

   The editor shows all the entries in the .dct file:



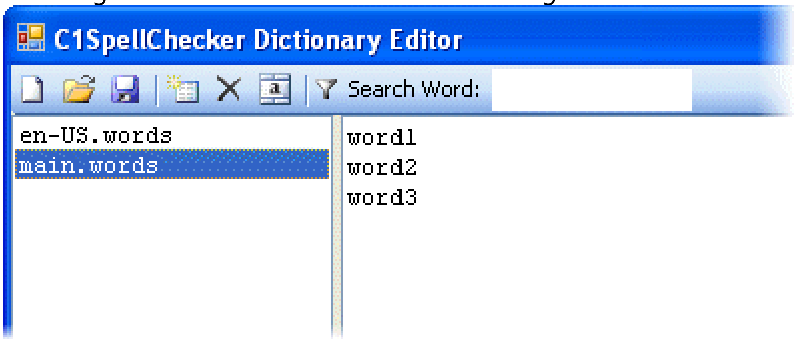
en-US.words

3Com
a
AA
AAA
AAAs
Aachen
Aalen
Aalto
Aar
aardvark
aardvarks
aardwolf
Aare
Aarhus
Aaron
Aarons

C:\Program Files\ComponentOne Studio.NET 2.0\bin\C1Spell_en-US.dct 118,042 words

The entries appear as items on the list box on the left. Selecting an entry shows its contents on the editor that appears on the right pane.

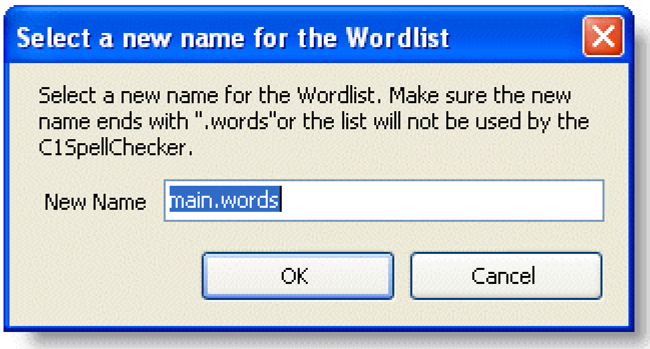
3. You can edit the contents of the .dct file using the following methods:
   - Typing or pasting words into the existing word list on the right.
   - Selecting the **Add Wordlist** button and creating a new word list to add:



en-US.words
main.words

word1
word2
word3

To rename the word list, select the **Rename Wordlist** button and change the default "main.words" name:



Select a new name for the Wordlist. Make sure the new name ends with ".words"or the list will not be used by the C1SpellChecker.

New Name: main.words

Tips: When creating a word list, note that case is important. Lowercase entries in word lists match regular words (for example, "word" matches "word", "Word", and "WORD"). Entries that start with an uppercase

> character do not match lowercase words and should be used for names (for example, "Paul" matches "Paul" and "PAUL", but *not* "paul").
> Also note that the words in word lists don't have to be in any specific order. The words will be sorted automatically when the file is saved.

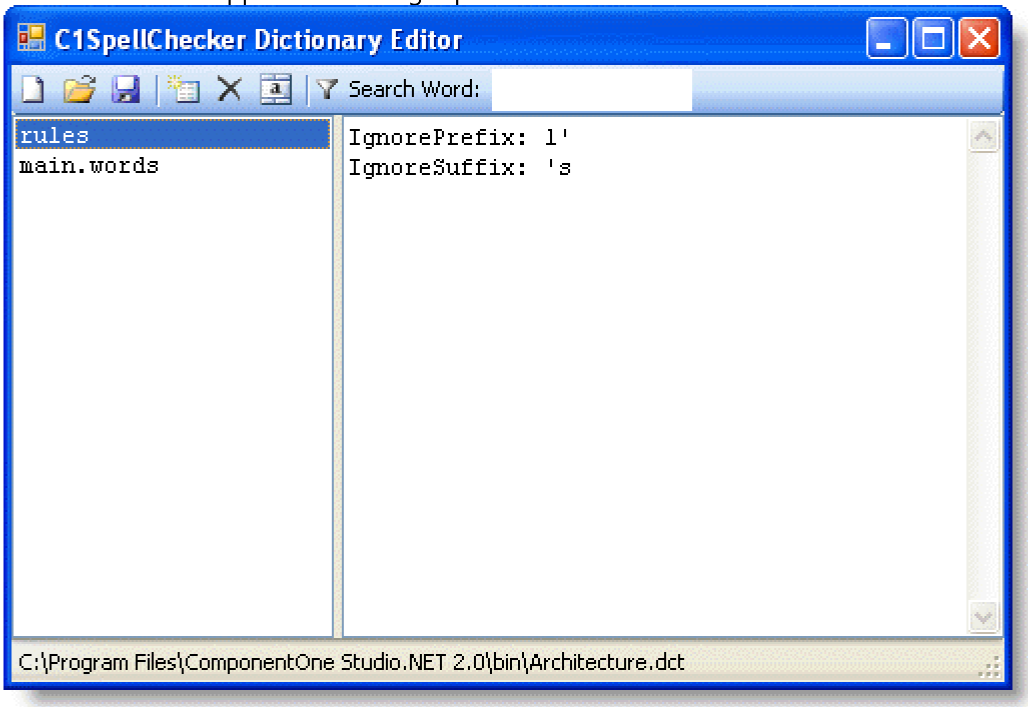
4. Once you are done making changes, save the file.

Creating a New DCT File

DCT files are simply zip files with one or more word lists. Each word list is an entry in the zip file, and its name must end with the ".words" extension (otherwise it will be ignored by the **C1SpellChecker**). Each word list is a UTF-8-encoded text file.

To create a new dictionary file (.dct), complete the following steps:

1. Double-click the **C1DictionaryEditor.exe** to open the **C1SpellChecker Dictionary Editor**.
2. Click the **New Dictionary** button . The **New Dictionary** dialog box appears.
3. Enter the name of the dictionary (for example, Architecture.dct) and press the **Save** button.
4. The editor creates rules and main.words items on the list box on the left. Selecting an entry shows its contents on the editor that appears on the right pane.


C1SpellChecker Dictionary Editor
Search Word:
rules
main.words
C:\Program Files\ComponentOne Studio.NET 2.0\bin\Architecture.dct

```
IgnorePrefix: l'
IgnoreSuffix: 's
```

The "rules" entry specifies rules to apply when spell-checking text in the dictionary language. For example, the new dictionary creates the following entries by default:

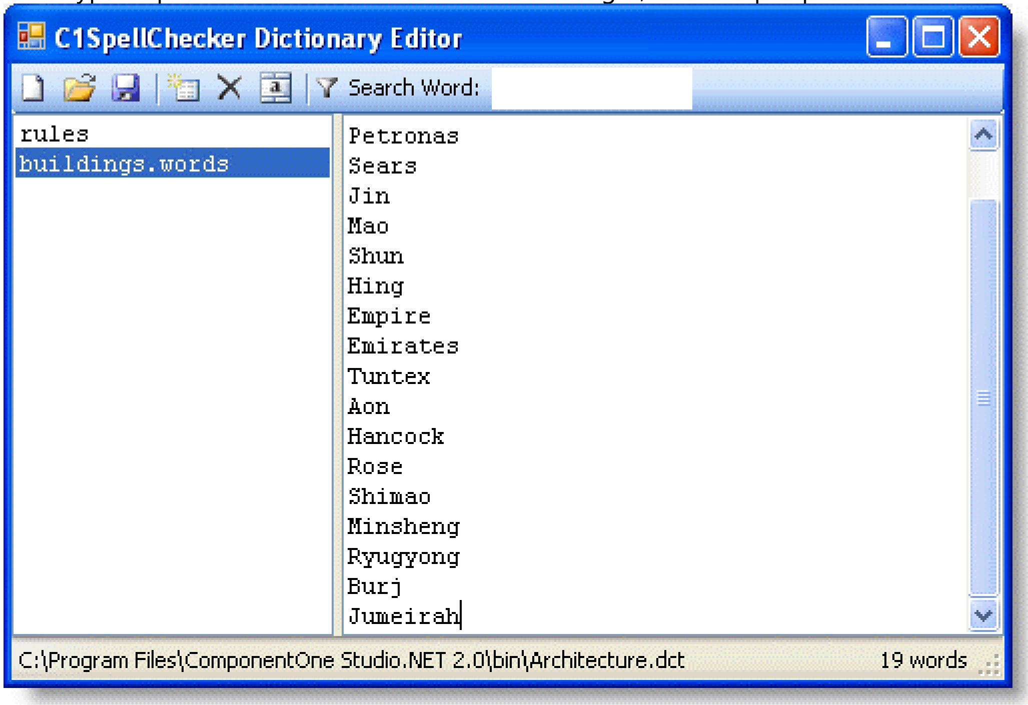
- IgnorePrefix: l'
- IgnoreSuffix: 's

For more information on rules, see the Types of Dictionary Files topic.

5. Select the **main.words** entry and press the **Rename Wordlist** button to open the **dialog box:**

   Rename the list, "buildings.words" for example, and click **OK**.

6. Then type or paste words into the word list on the right, for example: p



```
rules
buildings.words
```

```
Petronas
Sears
Jin
Mao
Shun
Hing
Empire
Emirates
Tuntex
Aon
Hancock
Rose
Shimao
Minsheng
Ryugyong
Burj
Jumeirah
```

C:\Program Files\ComponentOne Studio.NET 2.0\bin\Architecture.dct 19 words

Tips: When creating a word list, note that case is important. Lowercase entries in word lists match regular words (for example, "word" matches "word", "Word", and "WORD"). Entries that start with an uppercase character do not match lowercase words and should be used for names (for example, "Paul" matches "Paul" and "PAUL", but *not* "paul").
Also note that the words in word lists don't have to be in any specific order. The words will be sorted automatically when the file is saved.

Notice in the bottom-right corner of the dialog box the editor keeps count of the number of entries.

7. Once you are done making changes, save the file.

SpellChecker for WinForms Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos, which may make use of other development tools included with ComponentOne Studio.

Please refer to the pre-installed product samples through the following path:

Documents\ComponentOne Samples\WinForms

The following table provides a description for each sample.

Visual Basic and C# Samples

Sample	Description
AutoReplace	Shows how the AutoReplaceList property works. The AutoReplaceList contains a list of words and replacements. While the **C1SpellChecker** is monitoring a text box, it looks for matches against this list and automatically replaces them in the editor, as the user types.
C1SpellCheckerSpeed	Measures the C1SpellChecker performance when checking long documents. The sample has three buttons that spell-check the novel Moby Dick, the Declaration of Independence, and the King James Bible. After each document has been checked, the application shows the performance measurements.
CustomDictionary	Shows how to implement a custom spell dictionary class. The sample implements a CustomDictionary that accepts all words starting with the letter 'z' as correct.
CustomParser	Shows how to implement a custom spell-checking parser class. Custom parsers are classes that implement the ISpellParser interface.
CustomSpellDialog	Demonstrates how to implement a custom spell-checking modal dialog. This sample shows how to create your own spell dialog box and use that instead of the built-in one. Custom spell dialogs are Form objects that implement the ISpellDialog interface.
LocalizedSpellMenu	Shows how to customize the spelling context menu. The sample attached an event handler to the mouse down event of each control being spell-checked, then modifies the ContextMenuStrip for the controls by changing the menu items to Portuguese.
QuickStart	Shows how to use the **C1SpellChecker** component to check different types of controls: regular **TextBox**, **RichTextBox**, or **C1FlexGrid**.
SpellGrid	Shows how to implement a custom spell-checking modal dialog for a control (**C1FlexGrid**) that does not derive from **TextBoxBase**. This sample shows how to create a wrapper class that implements the ISpellCheckableEditor interface on behalf of a **C1FlexGrid** control.
TXTextSpellChecker	Shows how to implement as-you-type spell-checking with the TXTextControl. The TXTextControl is a very popular and powerful text editor control (http://www.textcontrol.com/). It supports advanced features like tables, doc and html file format import and export, pdf output, and more. The TXTextControl does not include a built-in spell checker. This sample shows how you can implement a control that derives from the TXTextControl and implements the interfaces required by the C1SpellChecker in order to support as-you-type spell-checking.
WebBrowserSpell	Demonstrates spell-checking editors that derive from the standard WebBrowser class. This sample shows underlining errors with red-wavy underlines as you type, with suggestions in context-sensitive menus, and also the dialog-based modal spell-check. The methods used are the same as the ones used with TextBoxBase controls: **SetActiveSpellChecking**, turns as-you-type spell checking on or off, and **CheckControl**, performs a dialog-based modal check

Sample	Description
	of the text in the control.

SpellChecker for WinForms Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio. By following the steps outlined in the help, you will be able to utilize the features of **C1SpellChecker**.

Each task-based help topic also assumes that you have created a new .NET project and have added the **C1SpellChecker** component to the Visual Studio Toolbox.

Creating a Custom Dictionary

In addition to the standard MainDictionary and UserDictionary dictionaries, **C1SpellChecker** also supports custom dictionaries. You can specify the custom dictionary with the CustomDictionary property.

For example, you can create a custom dictionary that accepts as correct any words that start with 'z'; therefore, any word that starts with 'z' will be correct.

1. From the Toolbox, add the **C1SpellChecker** component and **RichTextBox** control to your form.
   Note that the **C1SpellChecker** component will appear below the form, not on it.
2. Select the RichTextBox, and set the following properties:
   - **Dock** property to **Fill**.
   - **SpellChecking on C1SpellChecker1** property to **True**.
3. To specify the custom dictionary, add the following code:

To write code in Visual Basic

Visual Basic

```
Private _customDict As New MySpellDictionary()
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    C1SpellChecker1.CustomDictionary = _customDict
End Sub
Public Class MySpellDictionary
    Implements C1.Win.C1SpellChecker.ISpellDictionary
    Public Function Contains(ByVal word As String) As Boolean Implements
C1.Win.C1SpellChecker.ISpellDictionary.Contains
        Return word.StartsWith("z", StringComparison.InvariantCultureIgnoreCase)
    End Function
End Class
```

To write code in C#

C#

```
MySpellDictionary _customDict = new MySpellDictionary();

private void Form1_Load(object sender, EventArgs e)
{
    c1SpellChecker1.CustomDictionary = _customDict;
}
public class MySpellDictionary : C1.Win.C1SpellChecker.ISpellDictionary
{
    public bool Contains(string word)
    {
        return word.StartsWith("z", StringComparison.InvariantCultureIgnoreCase);
    }
}
```

Making Automatic Replacements as you Type

Use the AutoReplaceList property to spell-check a list of words, and if a match is found on the list then the misspelled word is replaced with the corresponding list entry as-you-type.

For example, you can monitor spelling on a Windows.Forms.RichTextBox control using this list. To see this feature in action, complete the following steps:

1. From the Toolbox, add the **C1SpellChecker** component and **RichTextBox** control to your form.

   Note that the **C1SpellChecker** component will appear below the form, not on it.

2. Select the RichTextBox, and set the following properties:
   - **Dock** property to **Fill**.
   - **SpellChecking on C1SpellChecker1** property to **True**.
3. Enter the following in your source code to declare the directive for the C1.Win.C1SpellChecker:

   To write code in Visual Basic

   Visual Basic
   ```
   Imports C1.Win.C1SpellChecker
   ```

   To write code in C#

   C#
   ```
   using C1.Win.C1SpellChecker;
   ```
4. Then double-click the RichTextBox and implement the following code in the body of the **RichTextBox1_TextChanged** event handler:

   To write code in Visual Basic

   Visual Basic
   ```
   ' build AutoReplace list
   C1SpellChecker1.AutoReplaceList.Clear()
   C1SpellChecker1.AutoReplaceList.Add("becuase", "because")
   C1SpellChecker1.AutoReplaceList.Add("cant", "can't")
   C1SpellChecker1.AutoReplaceList.Add("recieve", "receive")
   C1SpellChecker1.AutoReplaceList.Add("teh", "the")
   C1SpellChecker1.AutoReplaceList.Add("wont", "won't")
    ' activate as-you-type spell-checking on the RichTextBox
   C1SpellChecker1.SetActiveSpellChecking(RichTextBox1, True)
   ```

   To write code in C#

   C#
   ```
   // build AutoReplace list
   ```

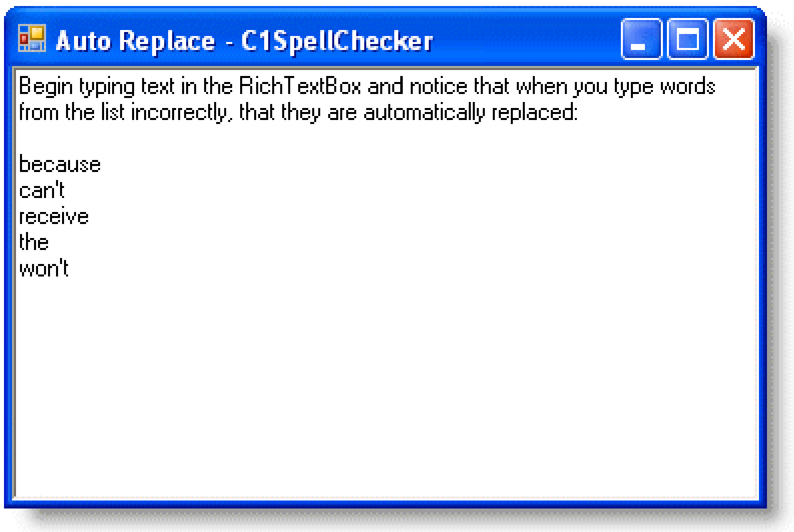
```
c1SpellChecker1.AutoReplaceList.Clear();
c1SpellChecker1.AutoReplaceList.Add("becuase", "because");
c1SpellChecker1.AutoReplaceList.Add("cant", "can't");
c1SpellChecker1.AutoReplaceList.Add("recieve", "receive");
c1SpellChecker1.AutoReplaceList.Add("teh", "the");
c1SpellChecker1.AutoReplaceList.Add("wont", "won't");
 // activate as-you-type spell-checking on the RichTextBox
c1SpellChecker1.SetActiveSpellChecking(richTextBox1, true);
```

This code builds the following list using the **AutoReplaceList.Add** method:

Misspelled word	Correction word
because	because
cant	can't
receive	receive
the	the
wont	won't

Then it activates as-you-type spell-checking on the RichTextBox control using the SetActiveSpellChecking method.

Run the application and observe the following:



Begin typing text in the RichTextBox and notice that when you type words from the list incorrectly, that they are automatically replaced:

because
can't
receive
the
won't

Setting the Spell Dialog Language

You can specify the language used in the **Spell Dialog** by setting the DialogLanguage property. For example, to set the language to **German** add the following code to your project:

To write code in Visual Basic

Visual Basic

```
C1SpellChecker1.Options.DialogLanguage = DialogLanguage.German
```

To write code in C#

C#

```
c1SpellChecker1.Options.DialogLanguage = DialogLanguage.German;
```

Note that the dialog language does not affect spelling. To change the language used for spelling, use the MainDictionary property to select a different spelling dictionary. For example, the following code sets the main dictionary to **German**:

To write code in Visual Basic

Visual Basic

```
C1SpellChecker1.MainDictionary.FileName = "C:\Program Files\ComponentOne Studio.NET
2.0\bin\C1Spell_de-DE.dct"
```

To write code in C#

C#

```
c1SpellChecker1.MainDictionary.FileName = "C:\\Program Files\\ComponentOne Studio.NET
2.0\\bin\\C1Spell_de-DE.dct";
```

Spell-Checking a C1FlexGrid Control

To spell-check other types of controls (a grid for example), you have to create a wrapper class that implements the ISpellCheckableEditor interface or the ISpellCheckableRichEditor interface. To do this, complete the following steps:

1. First, add the necessary controls and set some basic properties:
    1. From the Toolbox, add the **C1SpellChecker** component, **C1FlexGrid** control, and **Button** control to your form. Note that the **C1SpellChecker** component will appear in the component tray.
    2. Arrange the grid and button controls on the Form.
    3. Select the **Button** control and set its **Text** property to **Spell-Check the Grid** in the Properties window.
    4. Select the **C1FlexGrid** control and set its **Name** property to **_flex**.
    5. Select the **C1SpellChecker** control and set its **Name** property to **_spell**.
    6. Select the Form and set its **Name** property to **FlexGridForm**.
2. To specify the namespaces used in this example, add the following statements before any declarations in the Code Editor:

    To write code in Visual Basic

    Visual Basic

    ```
    Imports System.Data.OleDb
    Imports C1.Win.C1FlexGrid
    Imports C1.Win.C1SpellChecker
    ```

    To write code in C#

    C#

    ```
    using System.Data.OleDb;
    using C1.Win.C1FlexGrid;
    using C1.Win.C1SpellChecker;
    ```

3. To initialize the grid, double-click the Form and add the following code to the **FlexGridForm_Load** event. Note that you may have to change the connection string slightly, because it has a reference to the C1NWind.mdb database and that file might be in a different folder in your system:

   To write code in Visual Basic

   Visual Basic

   ```
   ' load data
   Dim sql As String = "select * from employees"
   Dim conn As String = "provider=microsoft.jet.oledb.4.0;data source=C:\Users\
   <User Name>\Documents\ComponentOne Samples\Common\C1NWind.mdb;"
   Dim da As New OleDbDataAdapter(sql, conn)
   Dim dt As New DataTable()
   da.Fill(dt)
    ' initialize grid
   _flex.Styles.Normal.WordWrap = True
   _flex.DataSource = dt
   Dim c As Column = _flex.Cols("Notes")
   c.Width = 350
   _flex.AutoSizeRows()
    ' hook up spell-checker when editing starts
   AddHandler _flex.StartEdit, AddressOf _flex_StartEdit
    ' use green underline here, just for fun
   _spell.Options.UnderlineColor = Color.DarkGreen
   /innovasys:widgetproperty>
   ```

   To write code in C#

   C#

   ```
   // load data
   string sql = "select * from employees";
   string conn = @"provider=microsoft.jet.oledb.4.0;data source=C:\Users\<User
   Name>\Documents\ComponentOne Samples\Common\C1NWind.mdb;";
   OleDbDataAdapter da = new OleDbDataAdapter(sql, conn);
   DataTable dt = new DataTable();
   da.Fill(dt);
    // initialize grid
   _flex.Styles.Normal.WordWrap = true;
   _flex.DataSource = dt;
   Column c = _flex.Cols["Notes"];
   c.Width = 350;
   _flex.AutoSizeRows();
    // hook up spell-checker when editing starts
   _flex.StartEdit += new RowColEventHandler(_flex_StartEdit);
    // use green underline here, just for fun
   _spell.Options.UnderlineColor = Color.DarkGreen;
   ```

4. Add the **StartEdit** event to the **C1FlexGrid** control and then add the following code inside the *flex*_StartEdit event. The **SetSpellChecking** method is used to provide as-your type spelling in the grid editor.

   To write code in Visual Basic

Visual Basic

```
' provide as-you-type spelling in the grid editor
Private Sub _flex_StartEdit(ByVal sender As Object, ByVal e As RowColEventArgs)
    Dim tb As TextBoxBase = TryCast(_flex.Editor, TextBoxBase)
    If tb IsNot Nothing Then
        _spell.SetSpellChecking(tb, True)
    End If
End Sub
```

To write code in C#

C#

```
// provide as-you-type spelling in the grid editor
void _flex_StartEdit(object sender, RowColEventArgs e)
{
    TextBoxBase tb = _flex.Editor as TextBoxBase;
    if (tb != null)
    {
        _spell.SetSpellChecking(tb, true);
    }
}
```

5. To spell-check the grid, double-click the **Button** control and add the following code to the **Button_Click** event:

To write code in Visual Basic

Visual Basic

```
' create spell-checkable wrapper for C1FlexGrid
Dim editor As New FlexGridSpeller(_flex, "Title", "Notes")
 ' spell-check
Dim errorCount As Integer = _spell.CheckControl(editor)
If errorCount > -1 Then
    Dim msg As String = String.Format("Spell-checking complete. {0} error(s)
found.", errorCount)
    MessageBox.Show(msg)
Else
    MessageBox.Show("Spell-checking cancelled.")
End If
```

To write code in C#

C#

```
// create spell-checkable wrapper for C1FlexGrid
FlexGridSpeller editor = new FlexGridSpeller(_flex, "Title", "Notes");
 // spell-check
int errorCount = _spell.CheckControl(editor);
 if (errorCount > -1)
  {
     string msg = string.Format("Spell-checking complete. {0} error(s) found.",
errorCount);
     MessageBox.Show(msg);
```

```
    }
else
{
    MessageBox.Show("Spell-checking cancelled.");
}
```

6. Add the following code to create a wrapper class that implements the ISpellCheckableEditor interface:

To write code in Visual Basic

Visual Basic

```
Public Class FlexGridSpeller
Implements ISpellCheckableEditor
'--------------------------------
#Region "** fields"
     Private _flex As C1FlexGrid
    ' grid being spell-checked
    Private _cols As Integer()
    ' columns to be spell-checked
    Private _row As Integer, _col As Integer
    ' cell being spell-checked (_row, _cols[_col])
    Private _selStart As Integer
    ' selection being checked within the cell
    Private _selLength As Integer
 #End Region
     '--------------------------------
#Region "** ctors"
     ' check some columns
    Public Sub New(ByVal flex As C1FlexGrid, ByVal ParamArray cols As String())
        ' save parameters
        _flex = flex
         ' create column list if needed
        If cols Is Nothing Then
            Dim list As New List(Of String)()
            For Each col As Column In flex.Cols
                If col.DataType.ToString() = "String" Then
                    list.Add(col.Name)
                End If
            Next
            cols = list.ToArray()
        End If
         ' convert column names to column indices
        _cols = New Integer(cols.Length - 1) {}
        For i As Integer = 0 To _cols.Length - 1
            Dim name As String = cols(i)
            If Not _flex.Cols.Contains(name) Then
                Throw New Exception("column not found: " + name)
            End If
            _cols(i) = _flex.Cols(name).Index
        Next
         ' scan cells until an error is found
        _row = -1
        _col = 0
        MoveNext()
    End Sub
     ' check all columns
    Public Sub New(ByVal flex As C1FlexGrid)
        Me.New(flex, Nothing)
    End Sub
 #End Region
     '--------------------------------
#Region "** object model"
     ' move on to the next cell
    Public Function MoveNext() As Boolean
        ' initialize or increment row/col position
        If _row < 0 Then
            ' initialize
            _row = _flex.Rows.Fixed
            _col = 0
        ElseIf _col < _cols.Length - 1 Then
            ' next column
            _col += 1
        Else
            ' next row
            _row += 1
            _col = 0
        End If
         ' return true if we still have valid cells
        Return _row < _flex.Rows.Count AndAlso _col < _cols.Length
    End Function
 #End Region
     '--------------------------------
#Region "** ISpellCheckableEditor"
     Public ReadOnly Property Control() As Control Implements
C1.Win.C1SpellChecker.ISpellCheckableEditor.Control
        Get
            Return _flex
        End Get
    End Property
    Public Property HideSelection() As Boolean Implements
C1.Win.C1SpellChecker.ISpellCheckableEditor.HideSelection
        Get
            Return False
        End Get
        Set(ByVal value As Boolean)
        End Set
    End Property
    Public Property Text() As String Implements
C1.Win.C1SpellChecker.ISpellCheckableEditor.Text
        Get
            Return _flex.GetDataDisplay(_row, _cols(_col))
        End Get
        Set(ByVal value As String)
```

```
            _flex(_row, _cols(_col)) = value
        End Set
    End Property
    Public Property SelectedText() As String Implements
C1.Win.C1SpellChecker.ISpellCheckableEditor.SelectedText
        Get
            Return Text.Substring(_selStart, _selLength)
        End Get
        Set(ByVal value As String)
            Dim t As String = Text
            t = String.Format("{0}{1}{2}", _
                Text.Substring(0, _selStart), _
                value, _
                Text.Substring(_selStart + _selLength))
            Text = t
        End Set
    End Property
    Public Property SelectionLength() As Integer Implements
C1.Win.C1SpellChecker.ISpellCheckableEditor.SelectionLength
        Get
            Return _selLength
        End Get
        Set(ByVal value As Integer)
            _selLength = value
        End Set
    End Property
    Public Property SelectionStart() As Integer Implements
C1.Win.C1SpellChecker.ISpellCheckableEditor.SelectionStart
        Get
            Return _selStart
        End Get
        Set(ByVal value As Integer)
            _selStart = value
        End Set
    End Property
    Public Sub [Select](ByVal start As Integer, ByVal length As Integer)
Implements C1.Win.C1SpellChecker.ISpellCheckableEditor.Select
        ' keep track of selection within the cell
        _selStart = start
        _selLength = length
         ' check that the cell being checked is selected
        _flex.[Select](_row, _cols(_col))
    End Sub
    Public Sub SelectAll()
        _selStart = 0
        _selLength = Text.Length
    End Sub
    Public Function HasMoreText() As Boolean Implements
C1.Win.C1SpellChecker.ISpellCheckableEditor.HasMoreText
        Return MoveNext()
    End Function
    Public Sub BeginSpell() Implements
C1.Win.C1SpellChecker.ISpellCheckableEditor.BeginSpell
     End Sub
    Public Sub EndSpell() Implements
C1.Win.C1SpellChecker.ISpellCheckableEditor.EndSpell
    End Sub
#End Region
End Class
```

To write code in C#

C#

```
public class FlexGridSpeller : ISpellCheckableEditor
{
    //-------------------------------
    #region ** fields
     // grid being spell-checked
    C1FlexGrid _flex;
    // columns to be spell-checked
    int[] _cols;
    // cell being spell-checked (_row, _cols[_col])
    int _row, _col;
    // selection being checked within the cell
    int _selStart;
    int _selLength;
     #endregion
    //-------------------------------
    #region ** ctors
     // check some columns
    public FlexGridSpeller(C1FlexGrid flex, params string[] cols)
    {
        // save parameters
        _flex = flex;
         // create column list if needed
        if (cols == null)
        {
            List list = new List();
            foreach (Column col in flex.Cols)
            {
                if (col.DataType == typeof(string))
                    list.Add(col.Name);
            }
            cols = list.ToArray();
        }
         // convert column names to column indices
        _cols = new int[cols.Length];
        for (int i = 0; i < _cols.Length; i++)
        {
            string name = cols[i];
            if (!_flex.Cols.Contains(name))
            {
                throw new Exception("column not found: " + name);
            }
            _cols[i] = _flex.Cols[name].Index;
        }

            // scan cells until an error is found
            _row = -1;
            _col = 0;
            MoveNext();
}
 // check all columns
public FlexGridSpeller(C1FlexGrid flex)
    : this(flex, null)
{
}
 #endregion
 //-------------------------------
#region ** object model
 // move on to the next cell
public bool MoveNext()
{
    // initialize or increment row/col position
    if (_row < 0)
    {
        // initialize
        _row = _flex.Rows.Fixed;
        _col = 0;
    }
    else if (_col < _cols.Length - 1)
    {
        // next column
        _col++;
    }
    else
    {
        // next row
        _row++;
        _col = 0;
    }

    // return true if we still have valid cells
    return _row < _flex.Rows.Count && _col < _cols.Length;
}
 #endregion
 //-------------------------------
#region ** ISpellCheckableEditor
 public Control Control
{
    get { return _flex; }
}
public bool HideSelection
```

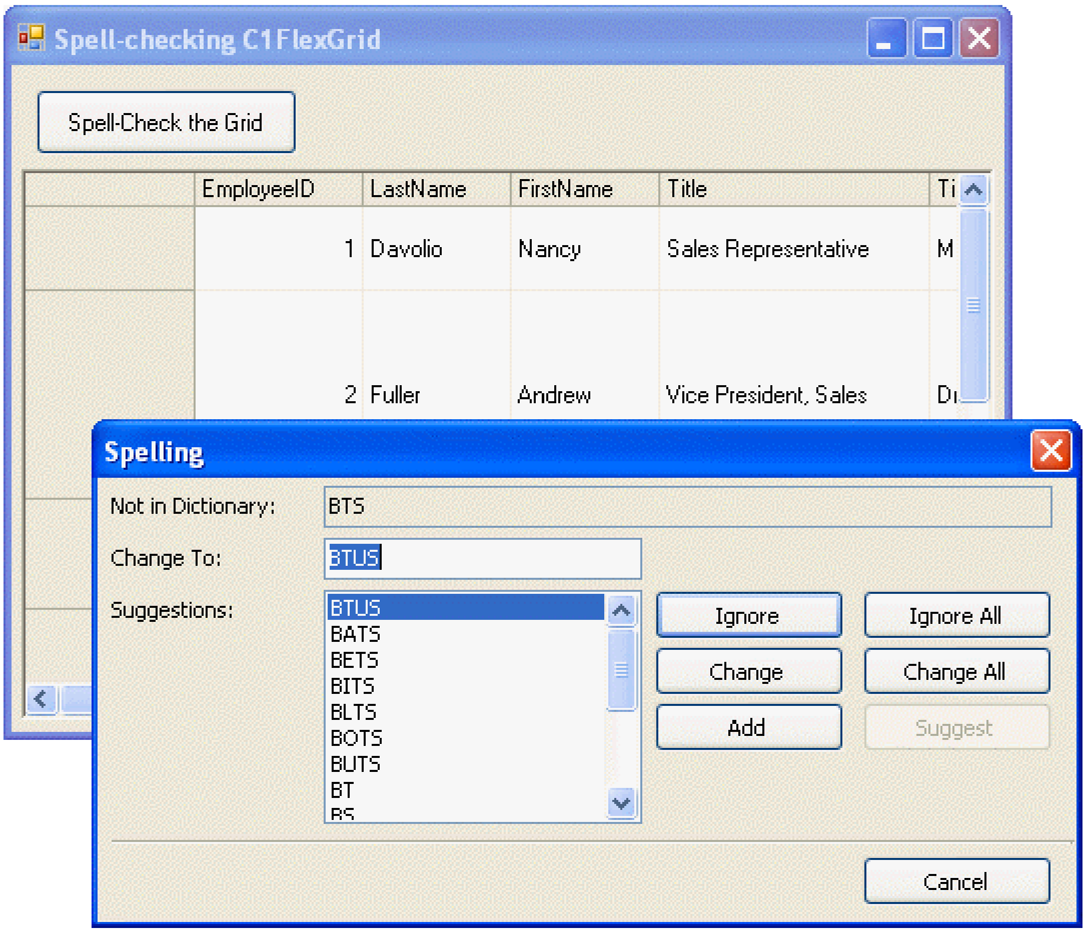
```
{
    get { return false; }
    set { }
}
public string Text
{
    get { return _flex.GetDataDisplay(_row, _cols[_col]); }
    set { _flex[_row, _cols[_col]] = value; }
}
public string SelectedText
{
    get { return Text.Substring(_selStart, _selLength); }
    set
    {
        string text = Text;
        text = string.Format("{0}{1}{2}",
            text.Substring(0, _selStart),
            value,
            text.Substring(_selStart + _selLength));
        Text = text;
    }
}
public int SelectionLength
{
    get { return _selLength; }
    set { _selLength = value; }
}
public int SelectionStart
{
    get { return _selStart; }
    set { _selStart = value; }
}
public void Select(int start, int length)
{
    // keep track of selection within the cell
    _selStart = start;
    _selLength = length;

    // check that the cell being checked is selected
    _flex.Select(_row, _cols[_col]);
}
public void SelectAll()
{
    _selStart = 0;
    _selLength = Text.Length;
}
public bool HasMoreText()
{
    return MoveNext();
}
public void BeginSpell()
```

```
        {
        }
        public void EndSpell()
        {
        }
        #endregion
}
```

Run the application and observe the following:

To spell-check the grid, press the **Spell-Check the Grid** button. The CheckControl method shows a **Spell Dialog** for the grid and returns the number of spelling errors found:



	EmployeeID	LastName	FirstName	Title	Ti
	1	Davolio	Nancy	Sales Representative	M
	2	Fuller	Andrew	Vice President, Sales	D

Not in Dictionary: BTS

Change To: BTUS

Suggestions: BTUS, BATS, BETS, BITS, BLTS, BOTS, BUTS, BT, BS