

---

ComponentOne

# SpellChecker for Silverlight

Copyright © 1987-2009 ComponentOne LLC. All rights reserved.

*Corporate Headquarters*  
**ComponentOne LLC**  
201 South Highland Avenue  
3<sup>rd</sup> Floor  
Pittsburgh, PA 15206 • USA

**Internet:** [info@ComponentOne.com](mailto:info@ComponentOne.com)  
**Web site:** <http://www.componentone.com>

**Sales**

E-mail: [sales@componentone.com](mailto:sales@componentone.com)  
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

**Trademarks**

ComponentOne SpellChecker for Silverlight and the ComponentOne SpellChecker for Silverlight logo are trademarks of ComponentOne LLC. ComponentOne is a registered trademark of ComponentOne LLC. All other trademarks used herein are the properties of their respective owners.

**Warranty**

ComponentOne warrants that the original media is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no cost to you. After 90 days, you may obtain a replacement for a defective media by sending your request and a check for \$25 (to cover postage and handling) to ComponentOne at the above address.

Except for the express warranty of the original media set forth herein, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was made public. ComponentOne does not warrant and therefore shall not be liable for any errors or omissions that the software documentation may contain. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not and shall not be held liable for any special, punitive, incidental, consequential, or any other damages that may result from your use of the software.

**Copying and Distribution**

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of the software by anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.



# Table of Contents

<b>ComponentOne SpellChecker for Silverlight</b> .....	<b>1</b>
<b>SpellChecker for Silverlight Concepts and Main Properties</b> .....	<b>3</b>
Spell Dictionaries.....	3
MainDictionary.....	3
UserDictionary.....	4
CustomDictionary.....	5
Spell-Checking Modes .....	5
Batch Mode.....	5
Modal Spell-Checking .....	5
As-you-type Spell-Checking.....	6
Spell-Checking Options.....	6
<b>Spell-Checking Dialog Boxes</b> .....	<b>9</b>
Using the Built-in Spell Dialog Box .....	9
Customizing the Built-in Spell Dialog Box.....	9
Using Custom Spell Dialog Boxes .....	10
<b>Spell-Checking Different Types of Controls</b> .....	<b>11</b>
<b>Spell Dictionary Format</b> .....	<b>13</b>



# ComponentOne SpellChecker for Silverlight

**C1SpellChecker** is a spell-checking engine that you can use to spell-check plain text or controls.

This is a Silverlight version of the **ComponentOne SpellChecker C1SpellChecker** component for **WinForms™**, a mature component that provides the fastest, easiest to use, and most flexible spell-checking solution available on the market. This is the same engine that drives **ComponentOne IntelliSpell™**, our add-in for Visual Studio that allows you to spell-check your applications for spelling mistakes in forms, web pages, and comments used to generate XML documentation.

The main features in **SpellChecker for Silverlight** are:

- **Efficiency:** On typical systems, **SpellChecker for Silverlight** is capable of spell-checking about 400,000 words per second. The spelling dictionaries are compressed and easy to maintain.
- **Ease of use:** All it takes to spell check a control is one call to the **CheckControl** method.
- **Flexibility:** **SpellChecker for Silverlight** provides several spell-checking services: you can spell-check strings and controls, and get suggestions for misspelled words. The interface-based architecture allows you to spell-check any controls that contain text, to implement custom spell dialog boxes, dictionaries, and text parsers. Finally, **SpellChecker for Silverlight** has a **SpellOptions** property that allows you to fine-tune the spell-checking options.
- **Multi-language support:** **SpellChecker for Silverlight** ships with over 20 international spell dictionaries, including English, Spanish, French, Italian, German, Portuguese, Dutch, Russian, Danish, Swedish and Greek.

You can download spell dictionaries for specific languages from this page:

<http://www.componentone.com/SuperPages/SpellCheckerDictionaries/>

This document describes how you can use the **C1.Silverlight.SpellChecker** assembly to add advanced spell-checking support to your Silverlight applications. If you find any errors in this document, or have suggestions for improving it, please write to [documentation@componentone.com](mailto:documentation@componentone.com).



# SpellChecker for Silverlight Concepts and Main Properties

The following topics outline **Spellchecker for Silverlight**'s concepts and primary properties.

## Spell Dictionaries

**SpellChecker for Silverlight** uses three types of dictionary to check text. Each dictionary is exposed as one of the three following properties: **MainDictionary**, **UserDictionary**, or **CustomDictionary**.

### MainDictionary

The main dictionary is a read-only, compressed word list that typically contains a few hundred thousand words. The main dictionary is typically loaded when the application starts, using the **LoadAsync** method as illustrated below:

```
public Page()
{
    InitializeComponent();

    // other initialization ...

    // load main dictionary
    SpellDictionary md = c1SpellChecker1.MainDictionary;
    md.LoadCompleted += md_LoadCompleted;
    md.LoadProgressChanged += md_LoadProgressChanged;
    md.LoadAsync("C1Spell_en-US.dct");
}
```

The **LoadAsync** method takes as a single parameter the URL of the spell dictionary, which is typically deployed as part of the application. The example above loads the **C1Spell\_en-US.dct** which is the US English spell dictionary that ships with **SpellChecker for Silverlight**. This dictionary contains about 120,000 words and is about 250k bytes in size.

The **LoadAsync** method loads the main dictionary asynchronously, so your application can start while the dictionary is downloaded from the server. When the download is complete, the **C1SpellChecker** component fires the **LoadCompleted** method which you can use to determine that the **C1SpellChecker** component is ready for work.

You can use the **MainDictionary.State** property to check at any time whether the main dictionary has been successfully loaded.

You can also load the main dictionary from a **Stream** object, so you can customize the loading process if you want.

Note that the **LoadAsync** method downloads the dictionary using a call to **WebClient.OpenReadAsync**. This call will fail if the server is not configured to serve the file extension used by the dictionary file. If this happens, you have a few easy ways to solve the problem:

- Rename the dictionary file to use an extension that is allowed by IIS. A good choice would be to use the “zip” extension (dictionary files really are zip files). This is the easiest option because it doesn’t require any changes to the server and no server-side code.
- If you don’t want to rename the file, configure IIS to allow serving “dct” files. This is done by adding the “dct” extension to the list of supported MIME types on the server. The drawback with this option is that it requires changes to the server configuration.
- Write a web service that reads the dictionary file and returns its contents as a byte array. Then open a **MemoryStream** using the array as the initial data and use the main dictionary’s **Load** method to load the dictionary from the **MemoryStream**. The drawback with this option is that it requires additional code on the client and on the server sides of the application.

## UserDictionary

The user dictionary is a read-write word list that contains terms defined by users, such as names and technical jargon (for example "ComponentOne", "Silverlight"). Users can add words to the custom dictionary while spell-checking by clicking the "Add" button in the spell dialog box. You can also add and remove words using code.

The user dictionary is typically stored in the applications isolated storage, rather than on the server. You can load and save the custom dictionary using the **LoadFromIsolatedStorage** and **SaveToIsolatedStorage** methods as shown below:

```
public Page()
{
    InitializeComponent();

    // other initialization ...

    // load main dictionary
    SpellDictionary md = c1SpellChecker1.MainDictionary;
    md.LoadCompleted += md_LoadCompleted;
    md.LoadProgressChanged += md_LoadProgressChanged;
    md.LoadAsync("C1Spell_en-US.dct");

    // load user dictionary
    UserDictionary ud = c1SpellChecker1.UserDictionary;
    ud.LoadFromIsolatedStorage("Custom.dct");

    // save user dictionary when app exits
    App.Current.Exit += App_Exit;
}
void App_Exit(object sender, EventArgs e)
{
    // save modified user dictionary into compressed isolated storage
    UserDictionary ud = c1SpellChecker1.UserDictionary;
    ud.SaveToIsolatedStorage("Custom.dct");
}
```

The code loads the user dictionary from isolated storage, and attaches an event handler to save any changes when the application exits. The user dictionary is typically small, and it is saved in compressed format, so it does not take up much storage space.

You can also load and save user dictionaries to **Stream** objects, so you can customize the persistence mechanism for the user dictionary if you want.

---

## CustomDictionary

Custom dictionaries are classes that derive from **C1Window** and implement the **ISpellDictionary** interface. This allows you to create dictionaries based on any custom logic that makes sense to your application. For example, you could create a dictionary that looks up words on the web and stores them in a cache, or that looks words up in a proprietary database.

Custom dictionaries are optional.

## Spell-Checking Modes

**SpellChecker** for **Silverlight** supports three modes of spell-checking: **Batch Mode**, **Modal Spell-Checking**, and **As-You-Type Spell-Checking**. The sections below provide information about each mode of spell-checking

### Batch Mode

The **CheckText**, **CheckWord**, and **GetSuggestions** methods check strings and get lists of errors and provide spelling suggestions. You can use these methods to spell check text that is not in a control. For example, you could spell-check text that is stored in a database.

The code below illustrates this mode:

```
// check some text
var someText = "this text contains two errors.";
var errors = c1SpellChecker1.CheckText(someText);
Debug.WriteLine("CheckText(\"{0}\") =", someText);
foreach (var error in errors)
{
    Debug.WriteLine("\t{0}, {1}-{2}",
        error.Text, error.Start, error.Length);
    foreach (string suggestion in
        c1SpellChecker1.GetSuggestions(error.Text, 1000))
    {
        Debug.WriteLine("\t\t{0}?", suggestion);
    }
}
```

### Modal Spell-Checking

The **CheckControlAsync** method will spell check controls that implement the **ISpellCheckableEditor** interface. The Microsoft **TextBox** control and the **C1RichTextBox** controls implement this interface out of the box. You can create your own classes to provide spell-checkable wrappers for other controls.

The code below illustrates this mode:

```
// show modal spell-checking
private void Button2_Click(object sender, RoutedEventArgs e)
{
    // hook up event handler
    c1SpellChecker1.CheckControlCompleted +=
        c1SpellChecker1_CheckControlCompleted;

    // spell-check a textbox
    c1SpellChecker1.CheckControlAsync(textBox1);

    // unhook the event handler
    c1SpellChecker1.CheckControlCompleted +=
        c1SpellChecker1_CheckControlCompleted;
}
void c1SpellChecker1_CheckControlCompleted(object sender,
    CheckControlCompletedEventArgs e)
{
    Debug.WriteLine("CheckControlCompleted: {0} errors found",
        e.ErrorCount);
    if (e.Cancelled)
        WriteLine("\t(cancelled...)");
}
```

This code shows a spell checking dialog box and highlights each spelling error, providing suggestions and allowing the user to fix or ignore each error. When the process is completed, the **CheckControlCompleted** event fires and provides feedback to the user.

### As-you-type Spell-Checking

This mode monitors a control while the user types, underlining errors with a red wavy line as in the Microsoft Word spell-checker. Right-clicking a misspelled word presents the user with a context menu containing suggestions.

This mode is not yet implemented in **SpellChecker for Silverlight**.

## Spell-Checking Options

The **Options** property allows you to fine-tune the spell checking process. It returns a **SpellOptions** object that contains the following properties:

- **DialogLanguage**: Customizes the language used in the built-in spell dialog box.
  - **Ignore**: Specifies whether to ignore words in upper-case, mixed-case, words that contain numbers, html tags, urls, etc.
  - **ActiveSpellingEnabled**: Gets or sets whether as-you-type spell-checking is enabled. This property isn't currently supported in **SpellChecker for Silverlight**.
  - **ShowSuggestionsInContextMenu**: Gets or sets whether the **C1SpellChecker** component should provide a context menu for misspelled words when as-you-type spell checking is enabled. This property isn't currently supported in **SpellChecker for Silverlight**.
-

- **MaxSuggestionsInContextMenu:** Gets or sets the maximum number of suggestions to be shown in the context menu associated with misspelled words. This property isn't currently supported in **SpellChecker for Silverlight**.
- **UnderlineColor:** Gets or sets the color used to underline misspelled words in as-you-type mode. This property isn't currently supported in the in **SpellChecker for Silverlight**.



# Spell-Checking Dialog Boxes

When you call the **CheckControlAsync** method, **SpellChecker for Silverlight** displays a dialog box that highlights each of the errors found in the control and allows the user to correct or ignore each error.

**SpellChecker for Silverlight** contains a built-in spell dialog box that is used by default. You can use the built-in dialog box as-is, customize it, or replace it with your own custom version.

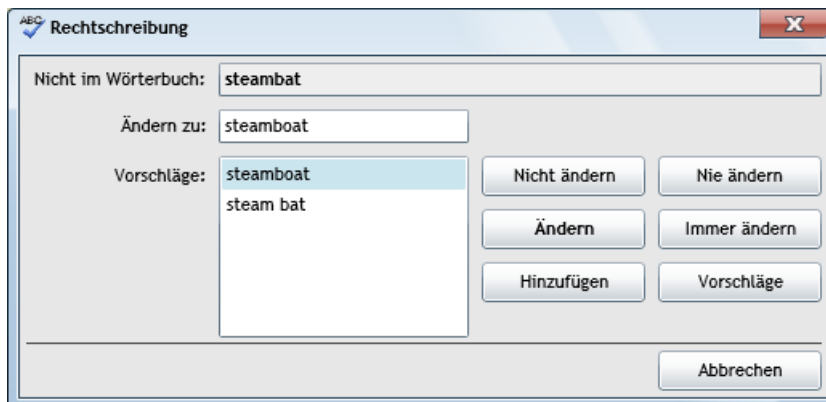
## Using the Built-in Spell Dialog Box

Using the built-in spell dialog box is easy. You don't have to do anything other than call the **CheckControlAsync** method.

The built-in dialog box has built-in localization with support for seven languages. The localization is controlled by the **DialogLanguage** property. The default setting, **Automatic**, causes the dialog box to be displayed in the language that corresponds to the **CurrentCulture** property. You can override that as shown below:

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

Here is the German version of the built-in spell dialog box:



Note that the dialog box language matches the application language and is independent of the current dictionary language. You could write an English application and use it to check German text for example.

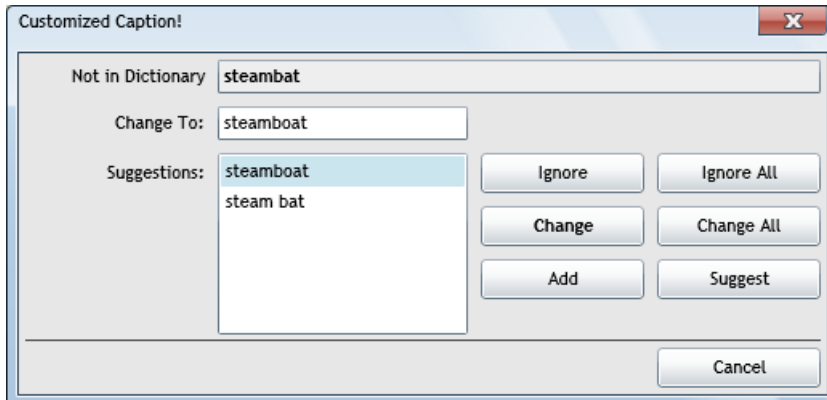
## Customizing the Built-in Spell Dialog Box

The built-in dialog box derives from **C1Window**. You can easily perform minor customizations to the dialog box by instantiating it, modifying its properties, and then passing the customized dialog box to the **CheckControlAsync** method.

For example, the code below modifies the caption of the built-in dialog box:

```
var dlg = new C1SpellDialog();
dlg.Header = "Customized Caption!";
c1SpellChecker1.CheckControlAsync(textBox1, false, dlg);
```

Here is the customized version of the built-in dialog box:



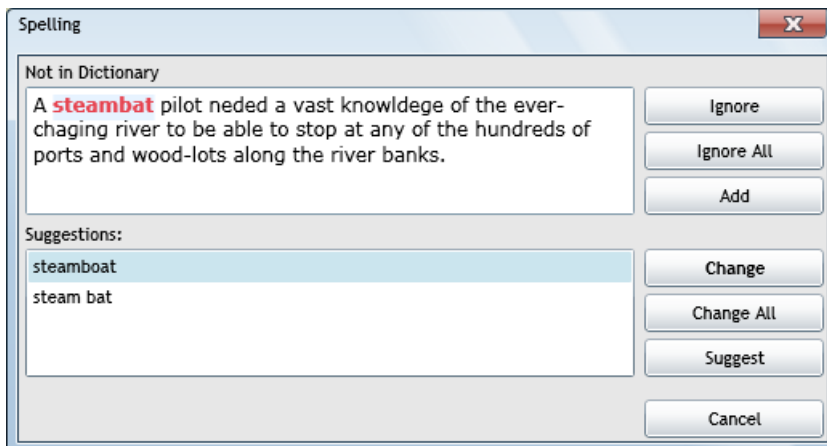
## Using Custom Spell Dialog Boxes

To use custom spell dialog boxes, you have to perform two tasks:

1. Create a class that derives from **CIWindow** and implements the **ISpellDialog** interface.
2. Use the overloaded version of the **CheckControlAsync** method that takes the **ISpellDialog** parameter.

To make the first task easier, we provide two dialog boxes that you can use as a starting point when creating your own dialog boxes. One of them is identical to the built-in dialog box, whereas the other is similar to the spell dialog box in Microsoft Word. You can find the source code in the "SpellCheckerSample" application that is installed in the "Samples" folder along with the product.

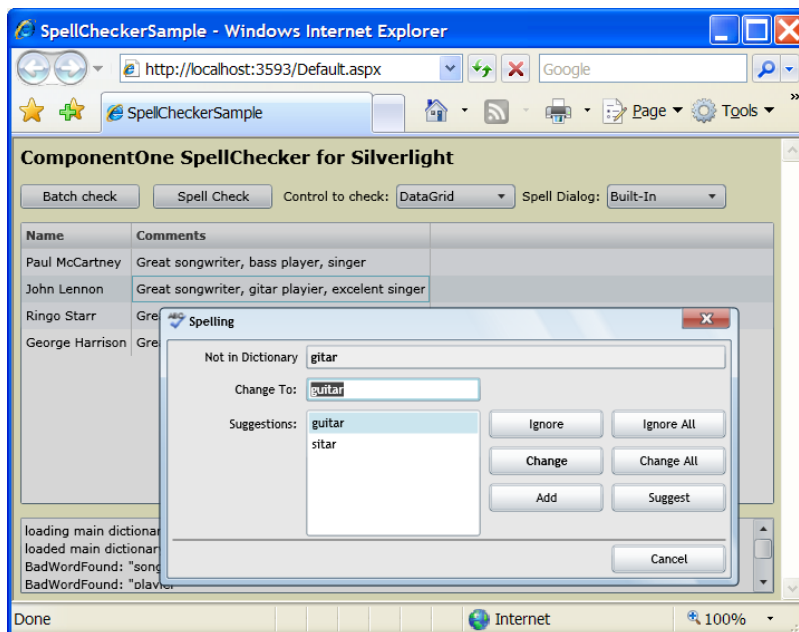
This is what the second custom dialog box looks like:



# Spell-Checking Different Types of Controls

**SpellChecker for Silverlight** can be used to spell check any control that contains text. This support is built-in for the **C1RichTextBox** and Microsoft **TextBox** controls. To spell-check other controls, you need to provide a wrapper class that implements the **ISpellCheckableEditor** interface.

For example, the "SpellCheckerSample" shows how you can spell-check a **DataGrid** control. It does this by creating a class called **DataGridSpellWrapper** that implements the **ISpellCheckableEditor** interface on behalf of the grid. The wrapper class enumerates the grid cells and exposes the text in each cell to the **C1SpellChecker** component. When an error is found, the cell is selected and the spell dialog box shows the error and suggestions as usual. The image below illustrates what the "SpellCheckerSample" looks like while spell-checking a **DataGrid**:





# Spell Dictionary Format

The main dictionaries are zip files with a .dct extension. The zip file may contain several word lists, each stored as a UTF-8-encoded text file containing lists of valid words, one per line. All such entries must have a ".words" extension.

Words that start with lowercase characters are assumed to be regular words, which can be used in lower-case or upper-case (e.g. "apple", "banana", "cherry"). Words that start with capitals are assumed to be proper names, and will be flagged as spelling mistakes if used in lowercase (e.g. "Albert", "Bernoulli", "Cantor").

The .dct 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 **SpellChecker for Silverlight** contains the following rules:

- **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 are tagged as spelling errors:

- h'amour (check 'h'amour': **incorrect**)
- x'amuor (check 'x'amuor': **incorrect**)

You can create and edit dictionary files using standard applications such as Notepad and WinZip, or you can use the **C1DictionaryEditor** application that ships with **SpellChecker for Silverlight**.