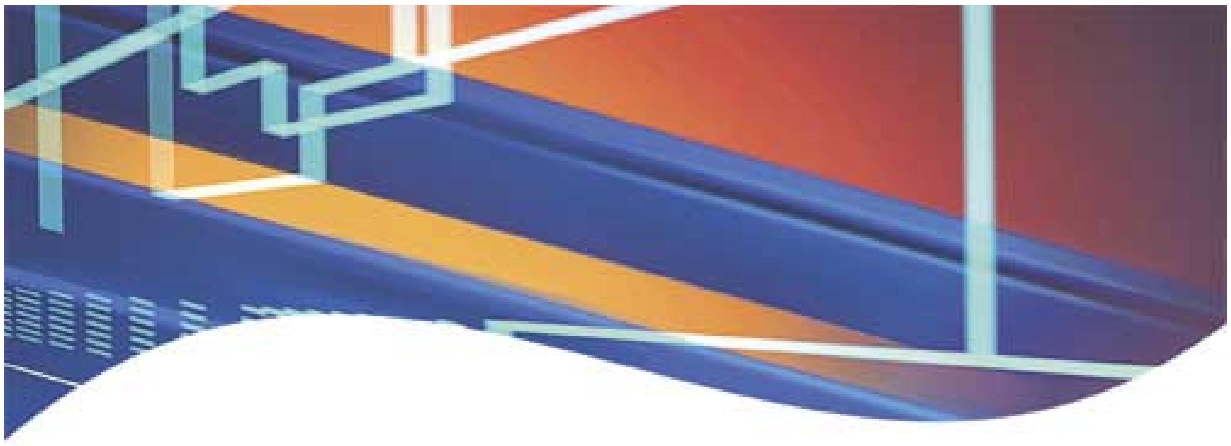


ComponentOne

VSSPELL™ 8.0

Spell-checking and Thesaurus Functionality
for your Visual Studio® Applications

ComponentOne

Corporate Headquarters
ComponentOne LLC
4516 Henry Street
Suite 500
Pittsburgh, PA 15213 USA

California Office
ComponentOne LLC
5900-T Hollis Street
Emeryville, CA 94608 USA

Internet: **info@ComponentOne.com**
Web site: **http://www.componentone.com**

Sales
E-mail: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)
Telephone: 1.888.228.4839 or 1.510.496.3240 (Emeryville, CA USA Office)

Technical Support
See *Technical Support* in this manual for information on obtaining technical support.

PRODUCED with ComponentOne Doc·To·Help®

Table of Contents

About ComponentOne VSSpell 8.0

Welcome to ComponentOne® **VSSpell**™! Now you can add spell-checking and thesaurus functionality to any Visual Basic® project using **VSSpell** or its companion, **VSThesaurus**. If you like **VSSpell**, make sure you check out our other award-winning products, ComponentOne VSREPORTS™, ComponentOne VSDOCX™, ComponentOne VS-OCX®, ComponentOne VSVIEW®, ComponentOne VSFlexGrid Pro®, and ComponentOne VSFORUM 2.0.

Our distribution policy is almost as innovative as our controls. We want every Visual Basic programmer to obtain a copy of **VSSpell** to try for as long as they wish. Those who like the product and find it useful may buy a license for a reasonable price. The only restriction is that unlicensed copies of **VSSpell** will display a ComponentOne banner every time they are loaded, to remind developers to license the product.

We are confident that you'll like **VSSpell**. If you have any suggestions or ideas for new features that you'd like to see included in a future version, or ideas about new controls, please call us or write:

Corporate Headquarters

ComponentOne LLC
4516 Henry Street
Suite 500
Pittsburgh, PA 15213 USA
412.681.4343
412.681.4384 (Fax)

California Office

ComponentOne LLC
5900-T Hollis Street
Emeryville, CA 94608 USA
888.228.4839
510.496.3240
510.595.2424 (Fax)

http://www.componentone.com

Overview

ComponentOne® **VSSPELL™** is a library consisting of **VSSpell** and **VSThesaurus** ActiveX controls. These controls allow you to add spell-checking and thesaurus functionality to any Visual Basic application.

Both components are available as 32-bit ActiveX controls.

Overview of the VSSpell control

Just place a **VSSpell** control on your form and you instantly have the power of ComponentOne® **VSSPELL™** 's extensive American English word dictionary. The **VSSpell** control features a built-in bad-word dialog box, the ability to generate suggestions for bad words, the ability to create new main dictionaries based on existing main dictionaries (such as those created in Microsoft Word), the ability to ignore all or change all bad words, and utilities to build and maintain dictionaries. Some of the features of the **VSSpell** control are:

-- Ability to link **VSSpell** to other controls to provide automatic spell checking as the user types.

-- Little code is needed to add spelling checking to your Visual Basic application.

-- Built-in bad-word dialog box.

-- Ability to check individual words, sentences or paragraphs.

-- Ability to generate alternate spellings for bad words.

-- Ability to create new dictionaries based on existing ones.

-- Support for custom dictionaries.

Overview of the VSThesaurus control

Place a **VSThesaurus** control on your form and gain the ability to provide users with a professional thesaurus similar to the one provided by Microsoft Word.

Some of the features of the **VSThesaurus** control are:

-- Little code is needed to add thesaurus functionality to your Visual Basic application.

-- Built-in automatic dialog box.

-- Ability to create new thesaurus files based on existing thesaurus files.

IMPORTANT: ComponentOne reserves the right to change the dictionary and thesaurus file formats with new versions of **VSSpell**. In these cases, a conversion utility will be supplied at no charge to our customers who upgrade.

Installing VSSpell and VSThesaurus

ComponentOne® **VSSpell**™ consists of the following major components:

-- **VSSpell** and **VSThesaurus** VBX and OCX Custom controls that are compatible with Windows development languages (such as Visual Basic).

-- Utilities to build and maintain Dictionaries.

-- Utilities to build and maintain Thesauruses.

To install **VSSpell** and **VSThesaurus**, use the **SETUP** utility provided on the installation compact disc (CD) or installation diskettes. When you are prompted, enter the registration key (found on the CD case or the diskette itself) exactly as it is printed and click REGISTER to complete the registration process. You may register any other ComponentOne products for which you have purchased a registration key at this time, as well.

VSSpell Setup Files

The setup program will copy the following VSSpell files to your computer:

CONVERTTO8.EXE	Converts to version 8.
SPELL8.OCX	VSSpell 32-bit custom control for Visual Basic.

VSSP_AE.DCT	American English 250,000+ word dictionary.
VSSPELL8.CHM	Help file used by the **VSSpell** and **VSThesaurus** custom controls.
DICTUTIL.EXE	Utility to create main dictionary files.
SAMPLES*.FRM	Visual Basic example form files.
SAMPLES*.MAK	Visual Basic example project files.
SAMPLES\BADWORD.FRM	A Visual Basic dialog box form for prompting the user when a bad word is encountered.
README.TXT	Release notes since the manual was printed.

VSThesaurus Setup Files

The setup program will copy the following **VSThesaurus** files to your computer:

THES8.OCX	**VSThesaurus** 32-bit custom control for Visual Basic.
VSTH_AE.THE	American English thesaurus.
THESUTIL.EXE	Utility to create main thesaurus files.
SAMPLES*.FRM	Visual Basic example form files.
SAMPLES*.MAK	Visual Basic example project files.
SAMPLES\BADWORD.FRM	A Visual Basic dialog box form for prompting the user when a bad word is encountered.
README.TXT	Release notes since the manual was printed.

System Requirements

- Microsoft Windows version 3.1 or higher
- A compatible Windows programming language (probably Visual Basic®)
- 1 MB hard disk space for storage of the software

- CD-ROM drive or 1.44 MB or 1.2 MB floppy drive

Installing Demonstration Versions

If you wish to try **VSSpell** or any of our other products, and do not have a registration key, use the SETUP.EXE file provided on the distribution CD. When prompted, leave the registration key box blank then press FINISH. The only difference between unregistered (demonstration) and registered (purchased) versions of our products is that registered versions will stamp every application you compile so a ComponentOne banner will not appear when your users run the applications.

Uninstalling

To uninstall **VSSpell**, use the UNSETUP.EXE utility provided on the installation CD. The uninstall program will remove all **VSSpell** files from your system.

Distributing your applications

When shipping your applications, we strongly suggest that you use a professional installation utility such as the Wise Installation System®, InstallShield®, or the setup wizard that ships with Microsoft Visual Basic 5.0 and later.

The following files should be shipped with your application:

- SPELL8.OCX
- THES8.OCX
- The main dictionary file (the default is VSSP_AE.DCT)
- The main thesaurus file (the default is VSTHE_AE.THE)

VSSpell and **VSThesaurus** are dependency-free controls. They do not require any additional DLLs in order to run.

End-User License Agreement

IMPORTANT-READ CAREFULLY: This End User License Agreement (this "**EULA**") contains the terms and conditions regarding your use of the SOFTWARE (as defined below). This EULA contains material limitations to your rights in that regard. You should read this EULA carefully and treat it as valuable property.

I. THIS EULA.

1. **Software Covered by this EULA.** This EULA governs your use of the ComponentOne, LLC ("**C1**") software product(s) enclosed or otherwise accompanied herewith (individually and collectively, the "**SOFTWARE**"). The term "SOFTWARE" includes, to the extent provided by C1: 1) any revisions, updates and/or upgrades thereto; 2) any data, image or executable files,

databases, data engines, computer software, or similar items customarily used or distributed with computer software products; 3) anything in any form whatsoever intended to be used with or in conjunction with the SOFTWARE; and 4) any associated media, documentation (including physical, electronic and on-line) and printed materials (the "**Documentation**").

2. **This EULA is a Legally Binding Agreement Between You and C1.** If you are acting as an agent of a company or another legal person, such as an officer or other employee acting for your employer, then "you" and "your" mean your principal, the entity or other legal person for whom you are acting. However, importantly, even if you are acting as an agent for another, you may still be personally liable for violation of federal and State laws, such as copyright infringement.

   This EULA is a legally binding agreement between you and C1. You intend to be legally bound to this EULA to the same extent as if C1 and you physically signed this EULA. By installing, copying, or otherwise using the SOFTWARE, you agree to be bound by the terms and conditions contained in this EULA. If you do not agree to all of the terms and conditions contained in this EULA, you may not install or use the SOFTWARE. If, for whatever reason, installation has begun or has been completed, you should cancel installation or un-install the SOFTWARE, as the case may be. (You may click on the "exit" button or its equivalent to immediately abort installation.) If you do not agree to all of these terms and conditions, then you must promptly return the SOFTWARE to the place of business from which you obtained it in accordance with any return policies of such place of business. Return policies may vary between or among resellers, and you must comply with your particular reseller's return policies as agreed at the point of purchase. If the place of business from which you purchased the SOFTWARE does not honor a complete refund for a period of thirty (30) days from the date of proof of purchase, then you may return the SOFTWARE directly to C1 for a period of thirty (30) days from the date of your purchase. To return the product directly to C1, you must obtain a C1 Return Authorization Number by contacting C1, and you must forward all items purchased, including the proof of purchase, directly to C1. The return must be postage-prepaid, and post-marked within thirty (30) days from the proof of purchase, time being of the essence. The return option to C1 is only available to the original purchaser of an unopened factory packaged item.

II. YOUR LICENSE TO DEVELOP AND TO DISTRIBUTE.

As provided in more detail below, this EULA grants you two licenses: 1) a license to use the SOFTWARE to develop other software products (the "**Development License**"); and 2) a license to use and/or distribute the Developed Software (the "**Distribution License**"). Both of these licenses (individually and collectively, the "**Licenses**") are explained and defined in more detail below.

1. **Definitions.** The following terms have the respective meanings as used in this EULA:

   "**Network Server**" means a computer with one or more computer central processing units (CPU's) that operates for the purpose of serving other computers logically or physically connected to it, including, but not limited to, other computers connected to it on an internal network, intranet or the Internet. "**Web Server**" means a type of Network Server that serves other computers more particularly connected to it over an intranet or the Internet.

   "**Developed Software**" means those computer software products that are developed by or through the use of the SOFTWARE. "**Developed Web Server Software**" means those Developed Software products that reside logically or physically on at least one Web Server and are operated (meaning the computer software instruction set is carried out) by the Web Server's central processing unit(s) (CPU). "**Developed Legacy Software**" means those Developed Software products that are not Developed Web Server Software, including, for example, stand-alone applications and applications accessed by a file server only. "**Redistributable Files**" means the SOFTWARE files or other portions of the SOFTWARE that are provided by C1 and are identified as such in the Documentation for distribution by you with the Developed Software. "**Developer**" means a human being or any other automated device using the SOFTWARE in accordance with the terms and conditions of this EULA.

   "**Developer Seat License**" means that each Developer using or otherwise accessing the programmatic interface or the SOFTWARE must obtain the right to do so by purchasing a separate End User License. "**Network Server CPU License**" means that a separate End User License must be purchased for each CPU operating the computer software at issue in the reference.

2. **Your Development License.** You are hereby granted a limited, royalty-free, non-exclusive right to use the SOFTWARE to design, develop, and test Developed Software, on the express condition that, and only for so long as, you fully comply with all terms and conditions of this EULA.

   The SOFTWARE is licensed to you on a Developer Seat License basis.

   The Developer Seat License means that you may perform a single install of the SOFTWARE for use in designing, testing and creating Developed Software by a single Developer on a single computer with a single set of input devices, so long as such computer is used only by one Developer. Conversely, you may not install or use the SOFTWARE on a computer that is a network server or a computer at which the SOFTWARE is used by more than one Developer. You may not network the SOFTWARE or any component part of it, where it is or may be used by more than one Developer unless you purchase an additional Development License for each Developer. You must purchase another separate license to the SOFTWARE in order to add additional developer seats, whether the additional developers are accessing the SOFTWARE in a stand-alone environment or on a computer network. If the SOFTWARE is used to create Developed Web Server Software, then you may perform a single install of the SOFTWARE for use in designing, testing and creating Developed Web Server Software by a single Developer on a single computer or Network Server, either with a single CPU only. An additional Network Server CPU License is required for each additional CPU on the single computer or Network Server, as the case may be, upon which the SOFTWARE and/or the Developed Web Server Software is installed and a separate license for each separate computer. That is, for Developed Web Server Software, if the computer or Network Server operating the SOFTWARE has 2 CPU's, then you would need to purchase one additional Network Server CPU License (beyond the one Developer Seat License granted in this EULA).

In all cases, you may not use C1's name, logo, or trademarks to market your Developed Software without the express written consent of C1; (b) you must include the following C1 copyright notice in your Developed Software documentation and/or in the "About Box" of your Developed Software, and wherever the copyright/rights notice is located in the Developed Software ("Portions Copyright © ComponentOne, LLC 1991-2002. All Rights Reserved."); (c) agree to indemnify, hold harmless, and defend C1, its suppliers and resellers, from and against any claims or lawsuits, including attorney's fees that may arise from the use or distribution of your Developed Software; (d) you may use the SOFTWARE only to create Developed Software that is significantly different than the SOFTWARE.

3. **Your Distribution License.**

a. License to Distribute Developed Legacy Software. Subject to the terms and conditions in this EULA, you are granted the license to use and to distribute Developed Legacy Software on a royalty-free basis, provided that the Developed Legacy Software incorporates the SOFTWARE as an integral part of the Developed Software in machine-language compiled format (customarily an ".exe", or ".dll", etc.). You may not distribute, bundle, wrap or subclass the SOFTWARE as Developed Software which, when used in a "designtime" development environment, exposes the programmatic interface of the SOFTWARE. You may distribute, on a royalty-free basis, Redistributable Files with Developed Legacy Software only.

b. License to Distribute Developed Web Server Software. Subject to the terms and conditions in this EULA, you are granted the license to use and to distribute Developed Web Server Software, provided that you must purchase one Network CPU License for each CPU operating the Developed Web Server Software (and/or Redistributable Files called or otherwise used directly by the Developed Web Server Software). You may purchase an additional Network CPU License by purchasing an additional separate End User License for the SOFTWARE; one Network CPU License is included in this EULA. Notwithstanding the foregoing, however, you may distribute or transfer, free of royalties, the Redistributable Files (and/or any Developed Legacy Software) to the extent that they are used separately on a single CPU on the client/workstation side of the network served by the Web Server.

4. **Specific Product Limitations.** Notwithstanding anything in this EULA to the contrary, if the license you have purchased is for any of the following products, then the following additional limitations will apply:

a. VS View Reporting Edition. VS View Reporting Edition includes at least one executable file listed as "VSRpt7.exe", known as "**Designer**." The file "VSRpt7.exe", or any upgrade or future versions of the Designer, are subject to the restrictions set forth in this EULA and may not be distributed with your Developed Software or in any other way.

b. VSForum. VSForum is intended to be installed on a Network Server. C1 grants to you the following rights to the SOFTWARE: a) Installation: You may install one copy of the SOFTWARE on a single Network Server; b) Use: When installed and initialized, the SOFTWARE creates a database file which contains the embodiment of a discussion forum (the database hereinafter referred to as the "**Forum**"). You may use the SOFTWARE to create one Forum on one Network Server only, which server may be connected at any point to an unlimited number of workstations or computers operating on one or more networks. You are specifically NOT LICENSED to create or operate multiple Forums. To create or to operate more than one FORUM, you must purchase one additional SOFTWARE license for each additional Forum.

c. Doc-to-Help. You may use Doc-To-Help to create online help, manuals or other documentation in electronic or printed format (the "**Output Documents**"). You may distribute and incorporate in such Output Documents those files identified in the documentation as Redistributable Files. Except for those specific Redistributable Files, you MAY NOT distribute the SOFTWARE, in any format, to others.

d. Studio Products. You may not share the component parts of the Studio Products licensed to you with other Developers, nor may you allow the use and/or installation of such components by other Developers.

5. **Updates/Upgrades; Studio Subscription.** Subject to the terms and conditions of this EULA, the Licenses are perpetual. Updates and upgrades to the SOFTWARE may be provided by C1 from time-to-time, and, if so provided by C1, are provided upon the terms and conditions offered at that time by C1 in its sole discretion. C1 may provide updates and upgrades to the SOFTWARE for free or for any charge, at any time or never, and through its chosen manner of access and distribution, all in C1's sole and complete discretion.

   C1 licenses certain of its separately-licensed products bundled together in a product suite, called the C1 "Studio" product line (the "**Studio Products**"). The exact separately-licensed products that are bundled into the Studio Products may change from time-to-time in C1's sole discretion. If the SOFTWARE is identified as a C1 "Studio" product, then the SOFTWARE is one of the Studio Products. The SOFTWARE and the Studio Products are revised from time-to-time (meaning, for example, revised with updates, upgrades and, in the case of Studio products, possibly changes to which specific products are included in the bundle). For you to be entitled to receive any such revisions to the SOFTWARE or the Studio Products, as the case may be, you must have a valid SOFTWARE license or a valid Studio subscription. The original purchaser of the SOFTWARE or of a Studio product receives a one-year subscription from the date of purchase of the SOFTWARE. After one year, the Studio subscription and/or the SOFTWARE license must be renewed to continue to be entitled to receive the SOFTWARE and/or the Studio Products revisions as the case may be.

6. **Serial Number.** Within the packaging of the SOFTWARE, a unique serial number (the "**Serial Number**") is included, which allows for the registration of the SOFTWARE. The Serial Number is subject to the restrictions set forth in this EULA and may not be disclosed or distributed either with your Developed Software or in any other way. The disclosure or distribution of the Serial Number shall constitute a breach of this EULA, the effect of which shall be the automatic termination and revocation of all the rights granted herein.

7. **Evaluation Copy.** If you are using an "evaluation copy" or similar version, specifically designated as such by C1 on its website or otherwise, then the Licenses are limited as follows: a) you are granted a license to use the SOFTWARE for a period of thirty (30) days counted from the day of installation (the "**Evaluation Period**"); b) upon completion of the Evaluation Period, you shall either i) delete the SOFTWARE from the computer containing the installation, or you may ii) contact C1 or one of its authorized dealers to purchase a license of the SOFTWARE, which is subject to the terms and limitations contained herein;

and c) any Developed Software may not be distributed or used for any commercial purpose.

III. INTELLECTUAL PROPERTY.

1. **Copyright.** You agree that all right, title, and interest in and to the SOFTWARE (including, but not limited to, any images, photographs, animations, video, audio, music, text, and "applets" incorporated into the SOFTWARE), and any copies of the SOFTWARE, and any copyrights and other intellectual properties therein or related thereto are owned exclusively by C1, except to the limited extent that C1 may be the rightful license holder of certain third-party technologies incorporated into the SOFTWARE. The SOFTWARE is protected by copyright laws and international treaty provisions. The SOFTWARE is licensed to you, not sold to you. C1 reserves all rights not otherwise expressly and specifically granted to you in this EULA.

2. **Backups.** You may either: (a) copy the SOFTWARE solely for backup or archival purposes; or (b) install the SOFTWARE on a single computer, provided you keep the original solely for backup or archival purposes. Notwithstanding the foregoing, you may not copy the Documentation.

3. **General Limitations.** You may not reverse engineer, decompile, or disassemble the SOFTWARE, except and only to the extent that applicable law expressly permits such activity notwithstanding this limitation.

4. **Software Transfers.** You may not rent or lease the SOFTWARE. You may transfer the SOFTWARE to another computer, provided that it is completely removed from the computer from which it was transferred. You may permanently transfer all of your rights under the EULA, provided that you retain no copies, that you transfer all the SOFTWARE (including all component parts, the media and printed materials, any dates, upgrades, this EULA and, if applicable, the Certificate of Authenticity), and that the recipient agrees to the terms and conditions of this EULA as provided herein. If the SOFTWARE is an update or upgrade, any transfer must include all prior versions of the SOFTWARE.

5. **Termination.** Without prejudice to any other rights it may have, C1 may terminate this EULA and the Licenses if you fail to comply with the terms and conditions contained herein. In such an event, you must destroy all copies of the SOFTWARE and all of its component parts.

6. **Export Restrictions.** You acknowledge that the SOFTWARE is of U.S. origin. You acknowledge that the license and distribution of the SOFTWARE is subject to the export control laws and regulations of the United States of America, and any amendments thereof, which restrict exports and re-exports of software, technical data, and direct products of technical data, including services and

Developed Software. You agree that you will not export or re-export the SOFTWARE or any Developed Software, or any information, documentation and/or printed materials related thereto, directly or indirectly, without first obtaining permission to do so as required from the United States of America Department of Commerce's Bureau of Export Administration ("**BXA**"), or other appropriate governmental agencies, to any countries, end-users, or for any end-uses that are restricted by U.S. export laws and regulations, and any amendments thereof, which include, but are not limited to, the following:

Restricted Countries: Restricted Countries currently include, but are not necessarily limited to Cuba, Iran, Iraq, Libya, Montenegro, North Korea, Serbia, Sudan, and Syria.

Restricted End-Users: Any End-User whom you know or have reason to know will use SOFTWARE or Developed Software in the design, development, or production of missiles and missile technology, nuclear weapons and weapons technology, or chemical and biological weapons. Any national of any of the Restricted Countries, wherever located, who intends to transmit or transport the SOFTWARE or Developed Software to one of the Restricted Countries.

Restricted End-Uses: Any use of SOFTWARE and Developed Software related to the design, development, or production of missiles and missile technology, nuclear weapons and weapons technology, or chemical and biological weapons.

These restrictions change from time to time. You represent and warrant that neither the BXA nor any other United States federal agency has suspended, revoked or denied your export privileges. C1 acknowledges that it shall use reasonable efforts to supply you with all reasonably necessary information regarding the SOFTWARE and its business to enable you to fully comply with the provisions of this Section. If you have any questions regarding your obligations under United States of America export regulations, you should contact the Bureau of Export Administration, United States Department of Commerce, Exporter Counseling Division, Washington DC. U.S.A. (202) 482-4811, http://www.bxa.doc.gov.

7. **U.S. Government Restricted Rights.** The SOFTWARE and documentation are provided with RESTRICTED RIGHTS. For solicitations issued before December 1, 1995, by the United States of America, its agencies and/or instrumentalities (the "**Government**"), other than the Department of Defense, the use, duplication or disclosure of the software and documentation provided to the Government under this EULA shall be subject to the RESTRICTED RIGHTS as set forth in subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at 48 CFR ch.1 52.227-19. For solicitations issued before September 29, 1995, by the Department of Defense, the use, duplication or disclosure of the

software and documentation provided under this EULA shall be subject to the RESTRICTED RIGHTS as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 48 CFR ch.2 252.227-7013. You will comply with any requirements of the Government to obtain such RESTRICTED RIGHTS protection, including without limitation, the placement of any restrictive legends on the SOFTWARE, and any license agreement used in connection with the distribution of the SOFTWARE. Manufacturer is ComponentOne, LLC, 4516 Henry Street, Suite 501, Pittsburgh, Pennsylvania 15213 USA. For solicitations issued by the Government on or after December 1, 1995 and the Department of Defense on or after September 29, 1995, the only rights provided in the software and documentation provided herein shall be those contained in this EULA. Under no circumstances shall C1 be obligated to comply with any Governmental requirements regarding the submission of or the request for exemption from submission of cost or pricing data or cost accounting requirements. For any distribution of the SOFTWARE that would require compliance by C1 with the Government's requirements relating to cost or pricing data or cost accounting requirements, you must obtain an appropriate waiver or exemption from such requirements for the benefit of C1 from the appropriate Government authority before the distribution and/or license of the SOFTWARE to the Government.

IV. WARRANTIES AND REMEDIES.

1. **Limited Warranty.** C1 warrants that the original media, if any, are free from defects for ninety (90) days from the date of delivery of the SOFTWARE. **EXCEPT AS OTHERWISE PROVIDED IN THE PRECEDING SENTENCE, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, C1 EXPRESSLY DISCLAIMS ANY WARRANTY FOR THE SOFTWARE, DOCUMENTATION AND ANYTHING ELSE PROVIDED BY C1 HEREBY AND C1 PROVIDES THE SAME IN "AS IS" CONDITION WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE AND DOCUMENTATION REMAINS WITH YOU. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS WHICH VARY FROM STATE TO STATE.**

2. **Limited Remedy.** C1's entire liability and your exclusive remedy under this EULA shall be, at C1's sole option, either (a) return of the price paid for the SOFTWARE; (b) repair the SOFTWARE through updates distributed online or otherwise in C1's discretion; or (c) replace the SOFTWARE with its reasonable, provided that you return the SOFTWARE in the same manner as provided in Section I.2 for return of the SOFTWARE for non-acceptance of this EULA. Any media for any repaired or replacement SOFTWARE will be warranted for the

remainder of the original warranty period or thirty (30) days, whichever is longer. THESE REMEDIES ARE NOT AVAILABLE OUTSIDE OF THE UNITED STATES OF AMERICA. **TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL C1 BE LIABLE FOR ANY DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFIT, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE, EVEN IF C1 HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES/JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES IN CERTAIN CASES, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.**

V. MISCELLANEOUS.

1. **This is the Entire Agreement.** This EULA (including any addendum or amendment to this EULA included with the SOFTWARE) is the final, complete and exclusive statement of the entire agreement between you and C1 relating to the SOFTWARE. This EULA supersedes any prior and contemporaneous proposals, purchase orders, advertisements, and all other communications in relation to the subject matter of this EULA, whether oral or written. No terms or conditions, other than those contained in this EULA, and no other understanding or agreement which in any way modifies these terms and conditions, shall be binding upon the parties unless entered into in writing executed between the parties, or by other non-oral manner of agreement whereby the parties objectively and definitively act in a manner to be bound (such as by continuing with an installation of the SOFTWARE, "clicking-through" a questionnaire, etc.) Employees, agents and other representatives of C1 are not permitted to orally modify this EULA.

2. **You Indemnify C1.** You agree to indemnify, hold harmless, and defend C1 and its suppliers and resellers from and against any and all claims or lawsuits, including attorney's fees, that arise or result from this EULA.

3. **Interpretation of this EULA.** If for any reason a court of competent jurisdiction finds any provision of this EULA, or any portion thereof, to be unenforceable, that provision of this EULA will be enforced to the maximum extent permissible so as to effect the intent of the parties, and the remainder of this EULA will continue in full force and effect. Formatives of defined terms shall have the same meaning of the defined term. Failure by either party to enforce any provision of this EULA will not be deemed a waiver of future enforcement of that or any other provision. Except as otherwise required or superseded by law, this EULA is governed by the laws of the Commonwealth of Pennsylvania, without regard to its conflict of laws principles. The parties consent to the personal jurisdiction

and venue of the Commonwealth of Pennsylvania, in the County of Allegheny, and agree that any legal proceedings arising out of this EULA shall be conducted solely in such Commonwealth. If the SOFTWARE was acquired outside the United States, then local law may apply.

Licensing

You are licensed to distribute the main dictionary file, main thesaurus file, the SPELL8.OCX, and the THES8.OCX free of royalties when you ship your application. You may include copies of the OCX files with as many copies of your application as you ship.

Groups of multiple developers may be interested in acquiring ComponentOne product site licenses. Please contact ComponentOne for details.

Technical Support

ComponentOne VSSpell is developed and supported by ComponentOne LLC, a company formed by the merger of APEX Software Corporation and VideoSoft. You can obtain technical support using any of the following methods:

ComponentOne Web site

The ComponentOne Web site at www.componentone.com provides a wealth of information and software downloads for VSSpell users, including:

- Descriptions of the various support options available through the ComponentOne Service Team.
- Answers to frequently asked questions (FAQ's) about our products, organized by functionality. Please consult the FAQ's before contacting us directly, as this can save you time and also introduce you to other useful information pertaining to our products.
- Free product updates, which provide you with bug fixes and new features.

Internet e-mail

For technical support through the Internet, e-mail us at:

support.vsspell@componentone.com

To help us provide you with the best support, please include the following information when contacting ComponentOne:

- Your ComponentOne product serial number.
- The version and name of your operating system.

- Your development environment and its version.

For more information on technical support, go to:

www.componentone.com/support

Peer-to-Peer newsgroup

ComponentOne also sponsors peer-to-peer newsgroups for users. ComponentOne does not offer formal technical support in this newsgroup, but instead sponsors it as a forum for users to post and answer each other's questions regarding our products. However, ComponentOne may monitor the newsgroups to ensure accuracy of information and provide comments when necessary. You can access the newsgroup from the ComponentOne Web site at www.componentone.com/newsgroups.

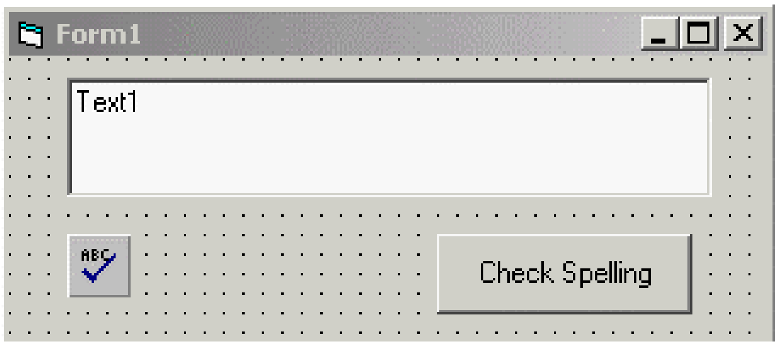
VSSpell QuickStart

This section will lead you through the creation of a simple Visual Basic project that uses the **VSSpell** control.

Begin by adding the **VSSpell** control to the Visual Basic toolbox. If you are not sure how to do this, reference your Visual Basic manual or help file on how to add custom controls. Next, add a **VSSpell** control to the form by double clicking on the **VSSpell** control button in the toolbox. Add a textbox and a command button to the form and set the following properties:

```
Command1.Caption = "Check Spelling"
Text1.Text = ""
```

Your form should look like the one shown below:



Main Dictionary File

Before any words can be checked, the **MainDictFile** property of the **VSSpell** control must be set to contain the complete path and filename of the main dictionary file. This may be done in design mode or at runtime. If a path and filename are not specified, the control will search for the default main dictionary filename (**vssp_ae.dct**) in the current directory, the Windows directory, the System directory, any directories in the MS-DOS Path environment variable, or any mapped network directories.

AutoLinkHwnd

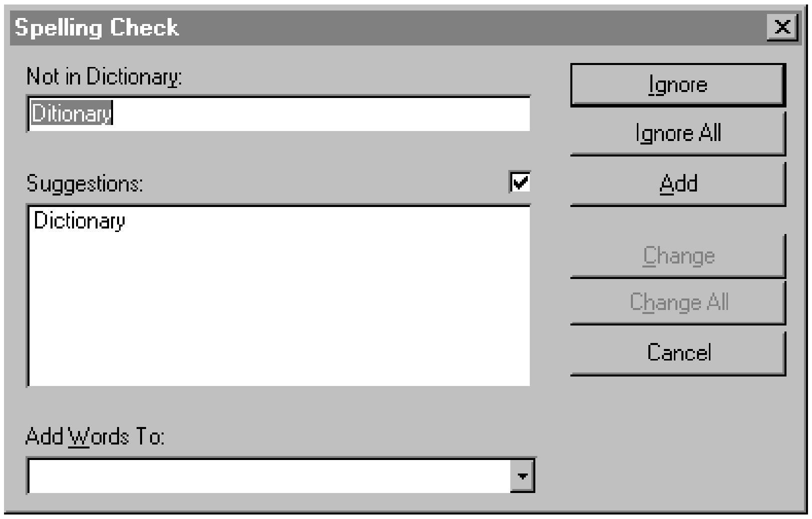
The **AutoLinkHwnd** property is used to link the **VSSpell** control to an external control, typically a regular text editor or a rich textbox (TextEdit or RichTextEdit controls). Set the **AutoLinkHwnd** property to the **hWnd** property of the control you want to spell check. The following code links the **VSSpell** control to the textbox and starts the spell checking when the Check Spelling button is pressed.

```
Private Sub Command1_Click()
  ' link spell check to the window
  VSSpell1.AutoLinkHwnd = Text1.hWnd
  ' you can start checking anywhere in the text
  VSSpell1.SelStart = 0
  ' start automatic spelling process
  VSSpell1.Start = True
End Sub
```

Note that you don't need to assign anything to the VSSpell's **Text** property. When **VSSpell** starts the spell-checking process, it will retrieve the text automatically from the Text1 control.

Bad-Word Dialog Box

Leave the **BadWordDialog** property set to the default setting, *vsspellEnglishDialog* (1). This causes a dialog box to appear when a bad word is encountered as shown below. This greatly reduces the code needed to handle bad words.



Suggestions

Leave the **Suggest** property set to the default setting, TRUE. This causes replacement suggestions for the bad word (if any are generated) to be displayed. The **Suggestion** event is fired each time a new suggestion is added to the **Suggestion** property array. The **BadWord** event is fired after the last **Suggestion** event for a word is fired. When the suggestions are displayed, the user has the following options:

-- Ignore the word this time only.

-- Change the word this time only.

-- Ignore the word and any other occurrences of the word.

-- Change the word and any other occurrences of the word.

-- Add the word to the custom dictionary.

-- Cancel the spell-checking process.

The **Suggestion** and **BadWord** events are fired whether or not the dialog box is used. However, if you are using the automatic dialog, there is no need to respond to the events, since the dialog box can handle the user's responses automatically.

That's all there is to it! You can now run your program. Type some words into the textbox and then click on the Check Spelling button. If the word is misspelled, the bad-word dialog box appears and the **BadWord** event is fired. If suggestions can be generated for the bad word, the **Suggestion** event is fired for each suggestion prior to the **BadWord** event being fired.

After the spell check is complete, the **Complete** event is fired.

Note: Any code following the setting of the **Start** property will begin executing after the **Complete** event is fired. For example, if you set the **CheckWord** or **Text** properties from code, any code following the statement that sets the **CheckWord** or **Text** property will not run until the **Complete** event is fired.

Other VSSpell Features:

Checking Multiple Words

To check multiple words, set the **Text** property with the words to check, then set the **Start** property to TRUE.

Checking a Single Word

To check a single word, set the **CheckWord** property to a string containing the word to be checked.

Dialog Box Option Button

Setting the **OptionBtnVisible** property to TRUE causes a customizable button to be shown on the dialog box. By default, the option button is not visible. The default caption for the button is "Options...", but it can be changed by setting the **OptionBtnCaption** property. Pressing the button fires the **OptionBtnClick** event.

Dialog Box Help Button

Setting the **HelpBtnVisible** property to TRUE causes a Help button to be shown on the dialog box. By default, the Help button is not visible. Pressing the Help button fires the **HelpBtnClick** event.

Property Arrays

The **IgnoreAll**, **ChangeAll**, and **ChangeAllTo** property arrays allow a word to be ignored or changed for all future occurrences of the word until the arrays are cleared (explained below). This is done by setting the respective properties **IgnoreAllWord** or **ChangeAllToWord** following a **BadWord** event, or when the user presses either the Ignore All or Change All buttons on the bad-word dialog box.

When a word is ignored for all occurrences of the word, the word being checked is added to the **IgnoreAll** property array. The **IgnoreAllCount** property returns the number of entries in the array. Setting the **ClearIgnoreAll** property to TRUE clears all entries from this array.

When a word is changed for all occurrences of the word, the word being checked is added to the **ChangeAll** property array and the word in the **ChangeAllToWord** property is added to the **ChangeAllTo** property array. There is only one count property since both arrays always contain the same number of entries. The **ChangeAllCount** property returns the number of entries in either array. Setting the **ClearChangeAll** property to TRUE clears all entries from both arrays.

All property arrays are zero-based. The first entry is 0, the second is 1, etc.

Invoking the **Clear** method clears all entries from the **IgnoreAll**, **ChangeAll**, and **ChangeAllTo** property arrays.

Custom Dictionary Files

The custom dictionary files are optional files. Each has a complete path and filename specified in the **CustomDictFile**, **CustomDictFile2**, **CustomDictFile3**, **CustomDictFile4**, and **CustomDictFile5** properties. A custom dictionary file is a text file with one word per line. Each line ends with a carriage return and a line-feed character. The words do not need to be sorted alphabetically.

The **CustomDictFile** properties can be set in design mode or at runtime. Setting these properties causes the custom dictionary files to be reread before the next spell check is performed.

Adding Words to a Custom Dictionary

Words can be added to a custom dictionary at runtime by setting the **AddCustomWord** property to the word to be added. Words are also added if the user presses the ADD button on the bad-word dialog box. Pressing the ADD button automatically adds the word to the custom dictionary. Use the **WhichCustomDict** property to tell **VSSpell** to which dictionary it should add the word.

Common Word Cache

A list of common words is built into **VSSpell** to help speed spell checking. This results in fewer accesses to the dictionary files. Setting the **CommonWordCache** property to FALSE disables this feature. Otherwise, set the property to a value that indicates which language cache will be used. Setting the **CommonWordCache** property to *vsspellEnglishCommonCache* (1) enables the American English common word cache. Setting the **CommonWordCache** property to *vsspellSpanishCommonCache* (2) enables the Spanish common word cache. The common word cache contains approximately 1,000 words and substantially increases the speed of the spell-checking process.

Property Groups

This section lists the main **VSSpell 8.0** properties and methods. The properties and methods are grouped according to function. For details on any property or method, refer to the main body of the documentation.

Initialization Properties

Set these properties before you start using the **VSSpell 8.0** control. You don't need to initialize properties with default values that are the ones you want to use.

MainDictFile	Always set this property before starting to use the VSSpell control, unless you know that the main dictionary file (VSSP_AE.DCT) is present in the Window directory or on the path.
IgnoreWithNumbers	Set these properties to specify types of words to ignore while spell checking.
IgnoreInUpperCase	Set these properties to specify types of words to ignore while spell checking.
IgnoreInMixedCase	Set these properties to specify types of words to ignore while spell checking.
Suggest	Set this property to specify whether or not the control should build a suggestion list and present it to the user when a bad word is found. You may inspect the list using the **Suggestion** and **SuggestionCount** properties.
BadWordDialog	Set this property to *vsspellNoDialog* (0) if you don't want the bad-word dialog box to appear every time a bad word is detected. You may also choose to display the dialog in a language other than English.
DialogTitle	Set these properties to customize the bad-word dialog.

DialogFont	Set these properties to customize the bad-word dialog.
DialogLeft	Set these properties to customize the bad-word dialog.
DialogTop	Set these properties to customize the bad-word dialog.

Perform Spell Checking

Use these properties and methods to spell check a control, a string, or a single word. Typically, you will use only one method depending on the type of application you are developing.

CheckWindow	Use this method to spell check the contents of a **TextEdit** or **RichTextEdit** control.
CheckTyping	Use this method to spell check a **TextEdit** or **RichTextEdit** control as the user types into it. Also set the **TypingErrorAction** to specify the action to be taken when an error is found, or trap the **TypingError** event to provide custom error handling.
Text / Start	Set these properties to spell check a string.
CheckWord	Set this property to spell check a single word.

Check Results

After the spell-checking process is done, or when **VSSpell** events are fired, you may want to inspect the results using these properties.

BadWordCount	Returns the number of bad words found while spell checking the document.
WordCount	Returns the number of words checked while spell checking the document.
WordsPerMinute	Returns the number of words checked per minute.
AverageWordLength	Returns the average length of the words that were spell checked.
CheckWord	Returns the word currently being spell checked.

Building and Maintaining Dictionary Files

The **VSSpell** distribution CD includes a utility program called **DICTUTIL.EXE** that allows you to create and maintain dictionary files. 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 **DICTUTIL.EXE** program to the ComponentOne directory you specify during the installation process.

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

1. Dump an existing dictionary file to a text file which you can then edit with any text editor.
2. Compile an existing text file into a new dictionary file.

To perform either function, follow these steps:

<u>To Dump an Existing Main Dictionary</u>

1) Click the button next to the "Dictionary File" field and select the dictionary file you want to dump into a text file.
2) Click the button next to the "Text File" field and select the name and location of the text file you want to create. If the file already exists, it will be overwritten.
3) Click the "Dump Dictionary to Text File" button.
4) Click the "x" button on the top right of the window to close the utility.

The text file generated by the **DICTUTIL** program contains one word per line. Each line ends with a carriage return and a line-feed character (**vbCrLf**). The last character of the file is an end-of-file character (ASCII code decimal 26 or Control Z).

Note that not all dictionaries may be dumped. If the dictionary was created with the Protection option enabled (the "Allow Others to Dump Dictionary" checkbox deselected) then you will not be able to dump the dictionary. If you plan to use the Protection option, make sure you save the words used to create the dictionary.

To Build a New Main Dictionary

1) Click the button next to the "Text File" field and select the text file containing the list of words that will be used to build the dictionary.
2) Click the button next to the "Dictionary File" field and select the name and location of the new dictionary file you want to create. If the file already exists, it will be overwritten.
3) Click the "Build Dictionary from Text File" button.
4) Click the "x" button on the top right of the window to close the utility.

When the build is complete, the new, main dictionary will contain the words from the text file.

When creating a text file that will be used to build a dictionary, remember these rules:

1. The text file must have one word per line.
2. Each line must end with a carriage return and a line-feed character (vbCrLf).
3. The last character of the file must be an end-of-file character (ASCII code decimal 26 or Control Z).
4. The words do not have to be in alphabetical order.
5. Case is not important.
6. Duplicate words will be detected and replaced with a single entry.

VSSpell Tutorial

This section will lead you through the creation of a Visual Basic project that uses the **VSSpell** control. The project shows how to use the **VSSpell** control to provide three types of spell checking:

1) **<u>Data-Entry Screens (As-you-type)</u>**: in this mode, the **VSSpell** control is linked to individual textboxes and monitors them as the user types. When errors are detected, the offending word is underlined and a beep sounds. The demo also provides an "AutoCorrect" option that will display a menu with suggestions when the user makes a typing error.

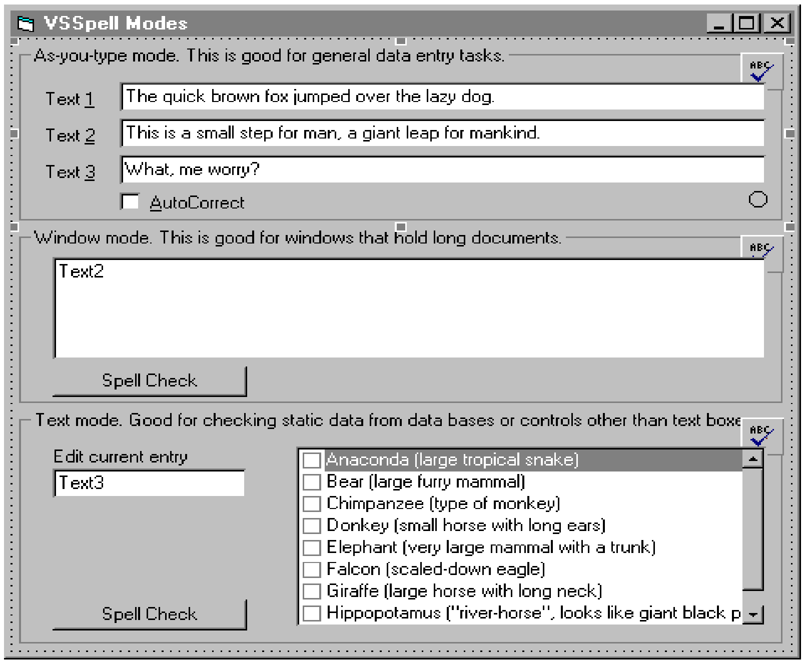
2) **<u>Long Documents</u>**: in this mode, the **VSSpell** control is invoked when the document is ready for checking. The demo uses the **CheckWindow** method and from then on the whole process is automatic. The text is retrieved from the control, a bad-word dialog is displayed whenever a bad word is found, and the changes made by the user through the dialog are applied to the control.

3) **<u>Plain or Arbitrary Text (no controls)</u>**: in this mode, the **VSSpell** control is used to check strings that come from arbitrary sources. The demo retrieves the strings from a ListBox control, but in practice the text comes from a database, text file, or any other source.

Step 1: Create the Main Form

Start a new Visual Basic project, add the **VSSpell** control to the Visual Basic toolbox, and add controls to create the main form. The form is divided in three sections, each consisting of a frame control with its own **VSSpell** control and other subordinate controls.

Here is how the form should look:



VSSpell Modes

As-you-type mode. This is good for general data entry tasks.

Text 1: The quick brown fox jumped over the lazy dog.

Text 2: This is a small step for man, a giant leap for mankind.

Text 3: What, me worry?

☐ AutoCorrect

Window mode. This is good for windows that hold long documents.

Text2

Spell Check

Text mode. Good for checking static data from data bases or controls other than text boxe

Edit current entry

Text3

Spell Check

- ☐ Anaconda (large tropical snake)
- ☐ Bear (large furry mammal)
- ☐ Chimpanzee (type of monkey)
- ☐ Donkey (small horse with long ears)
- ☐ Elephant (very large mammal with a trunk)
- ☐ Falcon (scaled-down eagle)
- ☐ Giraffe (large horse with long neck)
- ☐ Hippopotamus ("river-horse", looks like giant black p

The top panel will demonstrate As-you-type Spell Checking (Data Entry). It has the following controls:

Name	**Type**	**Function**
VSSpell1	VSSpell	Provides spell checking for this panel.
Text1(0)	TextEdit	This is where the user will type the text to be spell checked.
Text1(1)	TextEdit	This is where the user will type text to be spell checked.
Text1(2)	TextEdit	This is where the user will type text to be spell checked.
CheckBox1	CheckBox	Toggles auto-correct mode on and off.
Shape1	**Shape**	Used to show a red signal when the user makes a typing error.

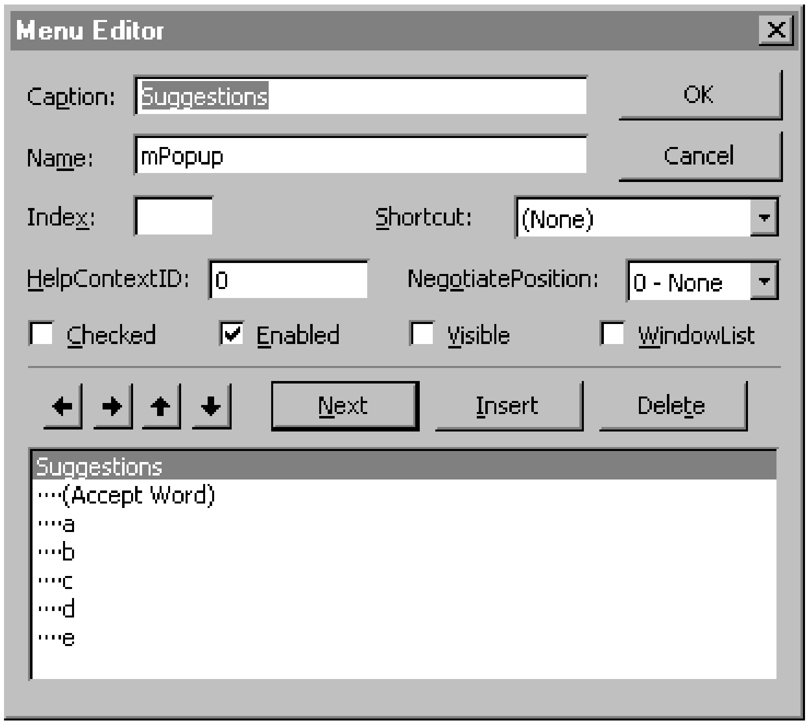
The middle panel will demonstrate Long Document Spell Checking. It has the following controls:

Name	**Type**	**Function**
VSSpell2	VSSpell	Provides spell checking for this panel.
Text2	TextEdit	This is where the user will type the text to be spell checked. This could also be a RichTextEdit control. To be able to see misspelled words better, set the HideSelection property to FALSE.
Command1	CommandButton	Starts spell checking the text in Text2.

The bottom panel will demonstrate Spell Checking Plain Text. It has the following controls:

Name	**Type**	**Function**
VSSpell3	**VSSpell**	Provides spell checking for this panel.
Text3	**TextEdit**	Used for changing ListBox entries.
List1	**ListBox**	Contains strings that will be spell checked. Set the Style property to 1 so the list will display checkboxes next to each item.
Command2	**CommandButton**	Spell check every ListBox item.

Also use the Visual Basic Menu Editor to create a single menu called *mPopup*, with an array of six sub-items. Make *mPopup* invisible, we'll display it using code. Here's how the menu should look in the Menu Editor window:



Caption: Suggestions
Name: mPopup
Index:
Shortcut: (None)
HelpContextID: 0
NegotiatePosition: 0 - None
☐ Checked ✓ Enabled ☐ Visible ☐ WindowList

Suggestions
····(Accept Word)
····a
····b
····c
····d
····e

Step 2: Initialize the Form

Double-click the form and add the following code to the *Form_Load* event:

```
Private Sub Form_Load()

    ' put some text in the large text window
    Text2 = Text1(0) & vbCrLf & Text1(1) & vbCrLf & Text1(2)
    ' select first item in listbox
    List1.ListIndex = 0
    Text3 = List1.List(List1.ListIndex)
    ' if the main dictionary is not on the system directory, set
it here
    VSSpell1.MainDictFile = App.Path & "\VSSP_AE.DCT"
    VSSpell2.MainDictFile = App.Path & "\VSSP_AE.DCT"
    VSSpell3.MainDictFile = App.Path & "\VSSP_AE.DCT"
End Sub
```

This routine initializes some text controls and the list box selection, then sets the VSSpell **MainDictFile** property.

Step 3: Provide As-you-type Spell Checking (top panel)

Double-click the first textbox and add the following code to the *Text1_GotFocus* event:

```
Private Sub Text1_GotFocus(Index As Integer)
'
' when moving the focus, start checking the control that has it
'
    VSSpell1.CheckTyping Text1(Index).hWnd
    VSSpell1.IntegerTag = Index
    Shape1.FillStyle = 1
End Sub
```

The code starts by connecting the **VSSpell** control to the textbox that has the focus. This is enough to provide basic As-you-type Spell Checking. It also saves the index of the currently active textbox into the **VSSpell1 IntegerTag** property, and makes the **Shape1** control transparent to indicate we haven't detected any errors yet.

By default, the **VSSpell** control will beep and underline misspelled words. Our demo will provide additional functionality. We will show a red sign when a bad word is detected and, optionally, display a list of suggestions. To do this, add the following code to the *VSSpell1_TypingError* event:

```
Private Sub VSSpell1_TypingError(ByVal SelStart As Long,
ByVal SelLength As Long, & _
    Cancel As Integer)
'
' error detected
'
    ' show red light to indicate an error was
    ' detected
    Shape1.FillColor = vbRed
    Shape1.FillStyle = 0 ' solid

    ' if AutoCorrect is not on, we're done
    If Check1.Value = 0 Then Exit Sub

    ' use the CheckWord property to build a list
    ' of suggestions
    VSSpell1.BadWordDialog = vsspellNoDialog
    VSSpell1.Suggest = True
    VSSpell1.CheckWord = VSSpell1.CheckWord

    ' no suggestions? just quit
    If VSSpell1.SuggestionCount = 0 Then Exit Sub

    ' if there is only one suggestion,
    ' assume it's OK
    ' replace automatically and cancel error
    If VSSpell1.SuggestionCount = 1 Then
        Text1(VSSpell1.IntegerTag).SelStart = SelStart
        Text1(VSSpell1.IntegerTag).SelLength = SelLength
        Text1(VSSpell1.IntegerTag).SelText =
VSSpell1.Suggestion(0)
        Cancel = True
        Shape1.FillColor = vbGreen ' the error
    ' has been fixed
        Exit Sub
    End If

    ' there are multiple suggestions,
    ' so build a menu
    Dim i As Integer
    For i = 1 To 5
        mSuggest(i).Visible = False
    Next
    mSuggest(0).Caption = VSSpell1.CheckWord & " (not in
dictionary)"
    For i = 1 To VSSpell1.SuggestionCount
        If i > 5 Then Exit For
        mSuggest(i).Caption = VSSpell1.Suggestion(i - 1)
        mSuggest(i).Visible = True
    Next
    ' menu is ready, display it at the
    ' current caret position
    ' the menu command will save the user's
    ' selection in the Tag property
    ' Note: the code subtracts the form's Left
    ' and Top properties from the
    ' CaretPosX and CaretPosY properties to
    ' convert screen into form coordinates.
    VSSpell1.Tag = ""
    PopupMenu mPopup, , VSSpell1.CaretPosX - Left,
VSSpell1.CaretPosY - Top
    If Len(VSSpell1.Tag) = 0 Then Exit Sub

    ' replace bad word with user selection
    Text1(VSSpell1.IntegerTag).SelStart = SelStart
    Text1(VSSpell1.IntegerTag).SelLength = SelLength
    Text1(VSSpell1.IntegerTag).SelText = VSSpell1.Tag
    ' error corrected!
    Cancel = True
    Shape1.FillColor = vbGreen

End Sub
```

This routine is pretty long, but it's fairly simple. If the "AutoCorrect" option is off, it simply shows the red shape to indicate that a word was misspelled and allows the **VSSpell1** control to provide the default user-feedback actions (beep and underline the offending word).

If "AutoCorrect" is on, the routine uses the **VSSpell1 CheckWord** property to build a list of suggestions. If **VSSpell1** cannot provide any suggestions, the routine returns immediately. If a single suggestion is provided, the code replaces the offending word with the suggestion automatically. This is probably not a great idea in practice, but it's pretty cool to watch as the control corrects words automatically. Finally, if many suggestions are available, the control assembles them into a pop-up menu and prompts the user to select one of the options. In the demo, we only use the first five suggestions.

To make the pop-up menu work, we need to implement the menu-handling function. All it needs to do is set the **VSSpell1 Tag** property to the word selected by the user:

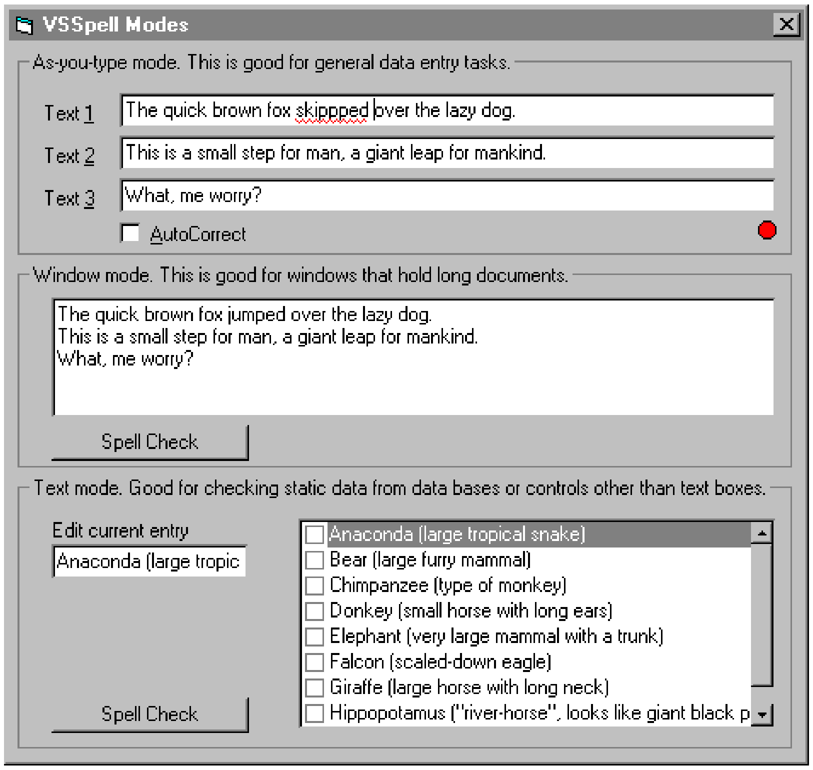
```
Private Sub mSuggest_Click(Index As Integer)

    ' index zero is whatever the user typed
    ' remember it so we don't pop up again
    ' on this word
    If Index = 0 Then
        VSSpell1.Tag = ""
        VSSpell1.IgnoreAllWord = True
        Exit Sub
    End If

    ' place corrected text in tag property
    VSSpell1.Tag = VSSpell1.Suggestion(Index - 1)

End Sub
```

We're almost done now. The only thing missing is the code to hide the red shape when the user types in a word that is correct. This can be done easily using the *VSSpell1_TypingOK* event.

```
Private Sub VSSpell1_TypingOK(ByVal SelStart As Long, ByVal
SelLength As Long)
'
' typed something right? erase red light.
'
    Shape1.FillStyle = 1
End Sub
```

That's it for the top pane on our demo. Save the project and run it. Start typing into the top three textboxes and you will see VSSpell's As-you-type Spell Checking in action.

Here's how the main form looks when an error is found:


VSSpell Modes
As-you-type mode. This is good for general data entry tasks.
Text 1
Text 2
Text 3
AutoCorrect
Window mode. This is good for windows that hold long documents.
Spell Check
Text mode. Good for checking static data from data bases or controls other than text boxes.
Edit current entry
Spell Check

Text 1: The quick brown fox skippped over the lazy dog.
Text 2: This is a small step for man, a giant leap for mankind.
Text 3: What, me worry?

The quick brown fox jumped over the lazy dog.
This is a small step for man, a giant leap for mankind.
What, me worry?

Edit current entry: Anaconda (large tropic

- Anaconda (large tropical snake)
- Bear (large furry mammal)
- Chimpanzee (type of monkey)
- Donkey (small horse with long ears)
- Elephant (very large mammal with a trunk)
- Falcon (scaled-down eagle)
- Giraffe (large horse with long neck)
- Hippopotamus ("river-horse", looks like giant black p

Step 4: Provide Long Document Spell Checking (middle panel)

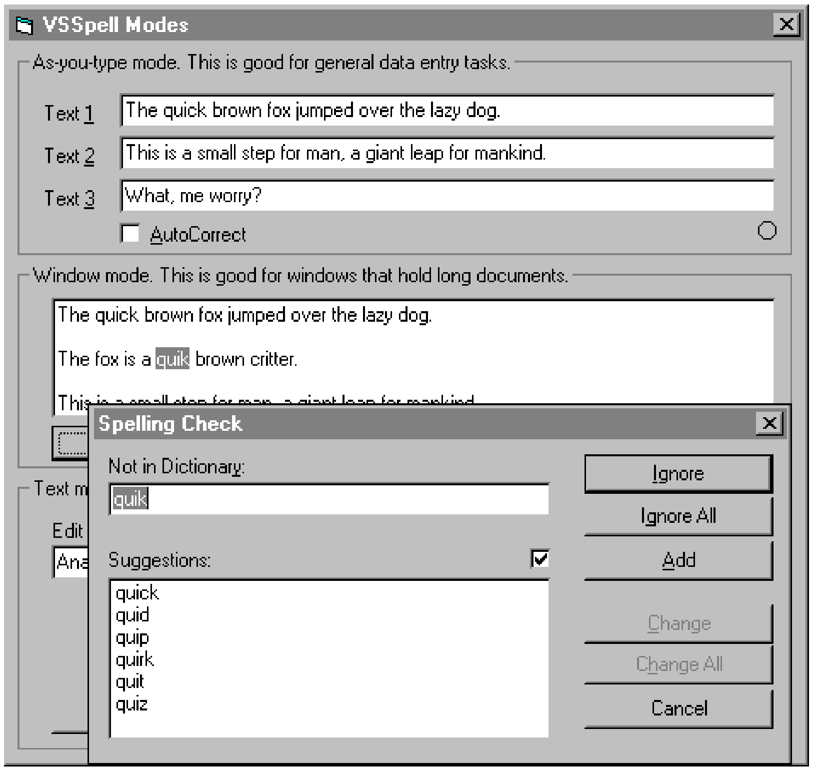
This step is really simple. Double-click the button on the middle panel and add the following code to the *Command1_Click* event:

```
Private Sub Command1_Click()
    ' this will check the whole window,
    ' displaying the bad-word dialog
    ' whenever a bad word is found and correcting
    ' words as needed
    VSSpell2.CheckWindow Text2.hWnd

    ' done, give the user some feedback
    If VSSpell2.BadWordCount > 0 Then
        MsgBox "Done spell checking. " & VSSpell2.BadWordCount & "
typing errors handled."
    End If
End Sub
```

All it takes to spell check the window is a single call to the **CheckWindow** method. We also show a message box after we're done spell checking, but this is optional.

If you think you will be spell checking really long documents (over 200K, say), you may also want to handle the *VSSpell2_Checking* event. This will fire every second while spell checking a long document so you can display a progress bar or cancel the spell-checking process.

That's it for the middle pane on our demo. Save the project and run it again. Type some text into the middle panel textbox and click the button below. If you make any mistakes, you will see a dialog box containing suggestions, as shown below:


VSSpell Modes

As-you-type mode. This is good for general data entry tasks.

Text 1: The quick brown fox jumped over the lazy dog.

Text 2: This is a small step for man, a giant leap for mankind.

Text 3: What, me worry?

AutoCorrect: unchecked

Window mode. This is good for windows that hold long documents.

The quick brown fox jumped over the lazy dog.

The fox is a quik brown critter.

Spelling Check

Not in Dictionary: quik

Suggestions: quick, quid, quip, quirk, quit, quiz

Step 5: Provide Spell Checking for Arbitrary Strings (bottom panel)

This part of the demo allows the user to change the contents of a list box and to spell check each item by clicking a button. In this case, the **VSSpell** control is not connected to any other controls. All spell checking is done through its **Text** property.

First of all, add the following two routines to connect the **Text3** and **List1** controls.

```
Private Sub List1_Click()
    Text3 = List1.List(List1.ListIndex)
End Sub
```

```
Private Sub Text3_Change()
    List1.List(List1.ListIndex) = Text3
End Sub
```

The first routine copies a selected item from **List1** into **Text3**. The second copies the contents of **Text3** into **List1** when the user makes any changes.

The final step is the routine that does the spell checking. It simply scans the **List1** control checking each list item and setting its selected state to TRUE if no spelling errors were found, or FALSE otherwise. Here is the code:

```
Private Sub Command2_Click()

    ' prepare control
    VSSpell3.BadWordDialog = vsspellNoDialog
    VSSpell3.Suggest = False
    ' spell check listbox items, select entries
    ' that look OK
    Dim i As Integer
    For i = 0 To List1.ListCount - 1
        VSSpell3.Text = List1.List(i)
        VSSpell3.CheckText
        If VSSpell3.BadWordCount > 0 Then
            List1.Selected(i) = False
        Else
            List1.Selected(i) = True
        End If
    Next
End Sub
```

That's it. This demo covers most aspects of the **VSSpell** control.

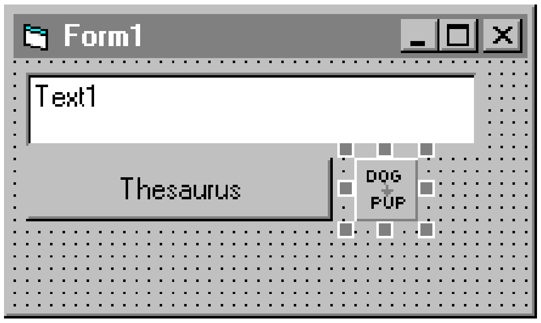
VSThesaurus QuickStart

This section will lead you through the creation of a Visual Basic project that uses the **VSThesaurus** control.

Begin by adding the **VSThesaurus** control to the Visual Basic toolbox. If you are not sure how to do this, reference your Visual Basic manual or Help file on how to add custom controls. Next, add a **VSThesaurus** control to the form by double clicking on the **VSThesaurus** control button in the toolbox. Add a textbox and a command button to the form and set the following properties:

```
 Command1.Caption = "Thesaurus"
Text1.Text=""
```

Your form should look like the one shown below:



Main Thesaurus File

Before any words can be checked, the **MainThesFile** property must be set to contain the complete path and filename of the main thesaurus file. This may be done in design time or at runtime. If a path and filename are not specified, the control will search for the default main thesaurus filename (**vsth_ae.the**) in the current directory, the Windows directory, the Windows System directory, any directories in the MS-DOS Path environment variable, or any mapped network directories.

Setting VSThesaurus to Check a Selected Word

To check a word selected in the textbox, set the **CheckWord** property to the selected string. The following code does this:

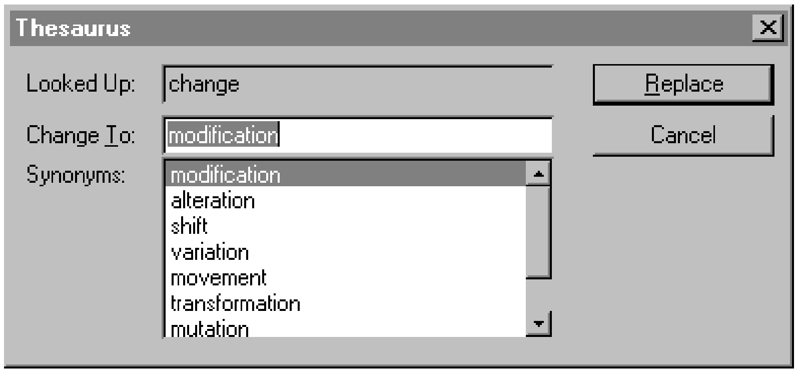
```
Private Sub Command1_Click()
  ' make sure a word is selected
  If Text1.SelLength = 0 Then Exit Sub
```

```
    ' trim any blank spaces from right end
    ' of the word
    Do While Text1.SelLength > 0
      If InStr(" ,.:;", Right(Text1.SelText, 1)) = 0 Then Exit Do
      Text1.SelLength = Text1.SelLength - 1
    Loop
    ' look it up (this may cause the Synonym
    ' dialog box to appear)
    VSThesaurus1.CheckWord = Text1.SelText
    ' replace word with user selection, if any
    If Len(VSThesaurus1.ChangedWord) > 0 Then
      Text1.SelText = VSThesaurus1.ChangedWord
    End If
  End Sub
```

Note: Once you set the **CheckWord** property from code, execution of your code is suspended until the **Complete** event is fired. In other words, any code following the statement that sets the **CheckWord** property will not run until the **Complete** event is fired.

Automatic Dialog Box

Leave the **AutomaticDialog** property set to its default value, *vsthesEnglishDialog* (1). This causes a dialog box (shown below) to show synonyms when a word is selected and the Thesaurus command button is pressed. The built-in dialog box greatly reduces the code needed to handle thesaurus checking.



Looked Up: change
Change To: modification
Synonyms: modification, alteration, shift, variation, movement, transformation, mutation

Synonyms are generated for a word that is found in the thesaurus. The **Suggestion** event is fired each time a new synonym is added to the **Synonym** property array. The **BadWord** event is fired if the word is not found in the thesaurus. The **Suggestion** event is not fired in this case.

The **Suggestion**, **BadWord**, and **Complete** events may be fired whether or not the dialog box is used. However, if you are using the automatic dialog, there is no need to respond to the events since the dialog box can handle the user's responses automatically.

You have completed all the necessary steps to incorporate thesaurus functionality. You can now run your program. Type some words in the textbox, select a word and then press the Thesaurus button. The Thesaurus dialog box will appear. If the word is not found in the thesaurus file, the **BadWord** event is fired. After the thesaurus check is complete, the **Complete** event is fired, whether the word is good or bad.

Other VSThesaurus Features:

Dialog Box Option Button

Setting the **OptionBtnVisible** property to TRUE (non-zero) causes a customizable option to be shown on the dialog box. By default, the option button is not visible. The default caption for the button is "Options...", but it can be changed by setting the **OptionBtnCaption** property. Pressing the button fires the **OptionBtnClick** event.

Dialog Box Help Button

Setting the **HelpBtnVisible** property to TRUE (non-zero) causes a Help button to be shown on the dialog box. By default, the Help button is not visible. Pressing the button fires the **HelpBtnClick** event.

VSSPELL8Lib Library

Description:	:-) ComponentOne VSSpell 8.0 Control
Library:	VSSPELL8Lib
File Name:	SPELL8.OCX
Help File:	VSSPELL8.CHM
GUID:	{08769121-33bd-11d3-bd95-b44cfe3a3c4b}
Control:	VSSpell

VSSpell Control

Before you can use a **VSSpell** control in your application, you must add the **SPELL8.OCX** file to your project. If you use the control in most of your VB projects, you may want to add it to Visual Basic's Autoload file.

To distribute applications you create with the **VSSpell** control, you must install and register it on the user's computer. The Setup Wizard provided with Visual Basic provides tools to help you do this. Please refer to the Visual Basic manual for details.

VSSpell Properties, Events, and Methods

All of the properties, events, and methods for the **VSSpell** control are listed in the following tables. Properties, events, and methods that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*). These are documented in later sections. For documentation on the remaining properties, see the Visual Basic documentation.

Properties

***AddBtnVisible**	***AddCustomWord**	***AutoLinkHwnd**
***AverageWordLength**	***BadWordCount**	***BadWordDialog**
***CaretPosX**	***CaretPosY**	***ChangeAll**
***ChangeAllCount**	***ChangeAllTo**	***ChangeAllToWord**
***ChangeWord**	***CheckSpelling**	***CheckWord**
***ClearChangeAll**	***ClearIgnoreAll**	***CommonWordCache**
***CustomDictFile**	***CustomDictFile2**	***CustomDictFile3**
***CustomDictFile4**	***CustomDictFile5**	***CustomDictListVisible**
***DialogChangedWord**	***DialogFont**	**DialogFont***
***DialogLeft**	***DialogTitle**	***DialogTop**
***DictionaryIsProtected**	***DictionaryLanguage**	***DictionaryVersion**
***DontCorrectText**	***HelpBtnVisible**	***IgnoreAll**

***IgnoreAllCount**	***IgnoreAllWord**	***IgnoreInMixedCase**
***IgnoreInUpperCase**	***IgnoreWithNumbers**	***IgnoreWord**
***IntegerTag**	***LastSpellError**	***MainDictFile**
***OptionBtnCaption**	***OptionBtnVisible**	***SelLength**
***SelStart**	***Start**	***Suggest**
***Suggestion**	***SuggestionCount**	***Text**
***TypingErrorAction**	***UnderlineColor**	***UnderlineStyle**
***WhichCustomDict**	***WordCount**	***WordsPerMinute**

Events

***BadWord**	***Changed**	***Checking**
***Complete**	***DialogAction**	***HelpBtnClick**
***OptionBtnClick**	***Suggestion**	***TypingError**
***TypingOK**		

Methods

***AddChangeAll**	***AddIgnoreAll**	***CheckText**
***CheckTyping**	***CheckWindow**	***Clear**

AddBtnVisible Property

Sets or returns whether the Add button is displayed on the bad-word dialog box.

Syntax

[form!]VSSpell.**AddBtnVisible**[= {True | False}]

Data Type

Boolean

Default Value

True

AddChangeAll Method

Adds a word to the **ChangeAll** and **ChangeAllTo** property arrays.

Syntax

[form!]VSSpell.**AddChangeAll** *Word* As String, *ChangeTo* As String

Remarks

This method allows you to programmatically add words to the **ChangeAll** and **ChangeAllTo** property arrays. This is equivalent to the user typing a replacement string for a bad word and clicking the **Change All** button on the bad-word dialog box.

You can clear the **ChangeAll** and **ChangeAllTo** property arrays using the **ClearChangeAll** property, but you cannot remove individual words from the arrays.

AddCustomWord Property

Adds the specified word to the custom dictionary specified by the **WhichCustomDict** property.

Syntax

[form!]VSSpell.**AddCustomWord** = *value* As String

Remarks

An error occurs if the **CustomDictFile** property or **CustomDictFile**(2-5) property is not set to a valid custom dictionary file name.

Data Type

String

AddIgnoreAll Method

Adds a word to the IgnoreAll property array.

Syntax

[form!]VSSpell.**AddIgnoreAll** *Word* As String

Remarks

This method allows you to programmatically add words to the **IgnoreAll** property array. This is equivalent to the user clicking the **Ignore All** button on the bad-word dialog box.

You can clear the **IgnoreAll** array using the **ClearIgnoreAll** property, but you cannot remove individual words from the array.

AutoLinkHwnd Property

Sets or returns the window handle (hWnd) of a control or window that will supply the text for spell checking.

Syntax

[form!]VSSpell.**AutoLinkHwnd**[= value As Long]

Remarks

The window handle supplied should belong to a **TextBox** or **RichTextBox** control.

After setting the **AutoLinkHwnd** property, start spell checking the control using the **Start** property. The text of the specified control or window will be placed in the VSSpell's **Text** property before the spell-checking process starts. Whenever an error is detected, the offending word will be automatically selected in the control, and changes made by the user will also be reflected in the control.

Set this property to zero to break the link from the control to a window.

Data Type

Long

AverageWordLength Property

Returns the average length of the words that were checked either using the **CheckWord** property or the **Text**/**Start** method of spell checking.

Syntax

val& = [form!]VSSpell.**AverageWordLength**

Data Type

Long

BadWord Event (VSSpell)

Fired when a bad word is encountered during a spell check.

Syntax

Private Sub VSSpell_**BadWord**(*Problem* As Integer)

Remarks

If suggestions are generated for the bad word, the **BadWord** event is fired after the final **Suggestion** event is fired. The problem parameter indicates the type of problem that caused the word to be bad. Currently, *NOTINDICT* is the only problem supported by the **BadWord** event.

When handling the **BadWord** event, you may retrieve the offending word from the **Text** property using the **SelStart** and **SelLength** properties, or read it directly from the **CheckWord** property. For example:

```
  Private Sub VSSpell1_BadWord(Problem As Integer)
      With VSSpell1
        Debug.Print "BadWord: "; Mid(.Text, .SelStart + 1,
.SelLength)
        Debug.Print "BadWord: "; .CheckWord
      End With
  End Sub
```

BadWordCount Property

Returns the number of bad words that were found during the last spell check.

Syntax

val& = [form!]VSSpell.**BadWordCount**

Remarks

This will be 1 if using the **CheckWord** property and a bad word is found. When using the **Start** property this will be the number of bad words that were encountered when **Start** was last set to a non-zero value.

Setting the **Start** property automatically resets this property.

Data Type

Long

BadWordDialog Property

Sets or returns whether the bad-word dialog will be used and the language that should be used to display it.

Syntax

[form!]VSSpell.**BadWordDialog**[= *val* As BadWordDialogConstants]

Remarks

Valid settings for the **BadWordDialog** property are:

Value	**Constant**	**Description**
0	vsspellNoDialog	No bad-word dialog will be displayed by the VSSpell control. The calling application may choose to display its own custom dialog.
1	vsspellEnglishDialog	The English bad-word dialog will be displayed.
2	vsspellSpanishDialog	The Spanish bad-word dialog will be displayed.
3	vsspellGermanDialog	The German bad-word dialog will be displayed.
4	vsspellFrenchDialog	The French bad-word dialog will be displayed.

Data Type

BadWordDialogConstants (Enumeration)

Default Value

vsspellEnglishDialog

CaretPosX Property

Returns the horizontal position of the caret on the edit window linked to the **VSSpell** control.

Syntax

[form!]VSSpell.**CaretPosX**[= value As Long]

Remarks

The caret position is returned in screen coordinates and expressed in twips.

This property is useful if you want to position user-interface elements such as pop-up menus or forms next to an offending word while spell checking. For example, to display a pop-up menu with a list of words, you would write code such as:

```
    PopupMenu mPopup, , VSSpell1.CaretPosX - Left,
VSSpell1.CaretPosY - Top
```

Note that the code subtracts the form's **Left** and **Top** properties from the values of **CaretPosX** and **CaretPosY**. This converts the coordinates from screen to form units, as required by Visual Basic's **PopupMenu** command.

Data Type

Long

CaretPosY Property

Returns the vertical position of the caret on the edit window linked to the **VSSpell** control.

Syntax

[form!]VSSpell.**CaretPosY**[= *value* As Long]

Remarks

See the **CaretPosX** property.

Data Type

Long

ChangeAll Property

Returns a specific word from the **ChangeAll** property array.

Syntax

val$ = [form!]VSSpell.**ChangeAll**(Index As Long)

Remarks

When a bad word is detected, the user has the option of specifying a replacement word and telling the control to change all occurrences of the word to the specified replacement word. In this case, the offending word is added to the **ChangeAll** array, and the specified replacement is added to the **ChangeAllTo** array.

The first word has index 0.

Data Type

String

ChangeAllCount Property

Returns the number of elements in the **ChangeAll** or **ChangeAllTo** string array.

Syntax

val& = [form!]VSSpell.**ChangeAllCount**

Data Type

Long

ChangeAllTo Property

Returns a specific word from the **ChangeAllTo** property array.

Syntax

val$ = [form!]VSSpell.**ChangeAllTo**(Index As Long)

Remarks

See the **ChangeAll** property.

Data Type

String

ChangeAllToWord Property

Setting this property adds the specified word to the **ChangeAllTo** property array.

Syntax

[form!]VSSpell.**ChangeAllToWord** = *value* As String

Remarks

When you assign a value to this property, the value is added to the **ChangeAllTo** property array, and the word being checked is added to the **ChangeAll** property array.

If the control is currently checking the **Text** property, the word indicated by the **SelStart** and **SelLength** properties is changed to the specified word.

Data Type

String

Changed Event (VSSpell)

Fired after the **Text** property changes.

Syntax

Private Sub VSSpell_**Changed**()

Remarks

This event is fired when the **Text** property changes while the text is being spell checked. This may occur in three situations:

1 The **ChangeWord** property was set or the Change button was pressed on the bad-word dialog box.

2 The **ChangeAllToWord** property was set or the Change All button was pressed on the bad-word dialog box.

3 The word being checked was found in the **ChangeAll** property array and was changed to the corresponding word in the **ChangeAllTo** property array. This is an automatic change without any user intervention.

ChangeWord Property

Setting this property changes the word being checked into the specified word.

Syntax

[form!]VSSpell.**ChangeWord** = *value* As String

Remarks

This property is designed to be used during the spell-checking process.

The word being checked may be read from the **CheckWord** property or from the **Text**, **SelStart**, and **SelLength** properties.

Data Type

String

Checking Event

Fired every second while spell checking to provide user feedback while checking long documents.

Syntax

Private Sub VSSpell_**Checking**(*Cancel* As Integer)

Remarks

This event is useful mainly when spell checking long documents (over 200k) to provide user feedback while the document is being spell checked.

You may stop the spell-checking process by setting the *Cancel* parameter to TRUE.

CheckSpelling Property

Sets or returns whether a spell check will be done.

Syntax

[form!]VSSpell.**CheckSpelling**[= {True | False}]

Remarks

Set this property to FALSE if you want to obtain statistics about the Text property, with no spell checking.

This will set the **AverageWordLength** and **WordCount** properties without changing the **Text** property.

Data Type

Boolean

Default Value

True

CheckText Method

Start spell checking the text in the **Text** property.

Syntax

[form!]VSSpell.**CheckText**

Remarks

Using this method is equivalent to setting the **Start** property to TRUE.

CheckTyping Method

Start spell checking a control as the user types.

Syntax

[form!]VSSpell.**CheckTyping** *hWnd* As Long

Remarks

The *hWnd* parameter passed to this method should correspond to an **EditBox** or **RichEditBox** control. After calling this method, **VSSpell** will start monitoring the target control as the user types into it. After each word is typed, **VSSpell** checks the word against the active dictionaries, the **IgnoreAll** array, and the **ChangeAll** array.

If a word is not found, **VSSpell** fires the **TypingError** event and executes the actions specified by the **TypingErrorAction** property. By default, **VSSpell** beeps and underlines the offending word with a wiggly red line, similar to the one used to highlight typing errors in Microsoft Word.

If the word is found in the **ChangeAll** array, it is automatically replaced with the corresponding entry in the **ChangeAllTo** array.

When a correct word is detected, **VSSpell** fires the **TypingOK** event. This event may be used to reset user-interface elements that alert the user of typing errors.

Note: While spell checking a control as the user types, the **Text** property is not used by **VSSpell**. This is done for efficiency since the contents of the control change constantly as the user types.

CheckWindow Method

Start spell checking the text in a given control.

Syntax

[form!]VSSpell.**CheckWindow** hWnd As Long

Remarks

Using this method is equivalent to setting **AutoLinkHwnd** property to a valid window handle and then setting the **Start** property to TRUE.

Usually, the window belongs to a **TextEdit** or **RichTextEdit** control, and is retrieved using the control's **hWnd** property. For example:

```
  VSSpell1.CheckWindow Text1.hWnd
```

CheckWord Property (VSSpell)

Sets a word to spell check or returns the word being checked.

Syntax

[form!]VSSpell.**CheckWord**[= *value* As String]

Remarks

If this property is set to a string containing more than one word, only the first word will be spell checked.

Read this property when handling a **BadWord** or **TypingError** event to retrieve the offending word.

Data Type

String

Clear Method

Clears the contents of all control properties.

Syntax

[form!]VSSpell.**Clear**

Remarks

This includes the **IgnoreAll**, **ChangeAll**, **ChangeAllTo**, and **Suggestion** property arrays as well as the **Text** property.

ClearChangeAll Property

Setting this property to TRUE clears the **ChangeAll** and **ChangeAllTo** property arrays.

Syntax

[form!]VSSpell.**ClearChangeAll** = {True | False}

Data Type

Boolean

ClearIgnoreAll Property

Setting this property to TRUE clears the **IgnoreAll** property array.

Syntax

[form!]VSSpell.**ClearIgnoreAll** = {True | False}

Data Type

Boolean

CommonWordCache Property

Sets or returns which cache of common words (if any) the control will use.

Syntax

[form!]VSSpell.**CommonWordCache**[= CommonWordCacheConstants]

Remarks

The settings for the **CommonWordCache** property are described below:

Value	**Constant**	**Description**
0	vsspellNoCommonCache	Do not use any common word cache.
1	vsspellEnglishCommonCache	Use the English common word cache.
2	vsspellSpanishCommonCache	Use the Spanish common word cache.

Using a common word cache greatly increases the speed of the control while spell checking long documents.

Data Type

CommonWordCacheConstants (Enumeration)

Default Value

vsspellEnglishCommonCache (1)

Complete Event (VSSpell)

Fired after the spell check has been completed.

Syntax

Private Sub VSSpell_**Complete**(*Cancelled* As Integer)

Remarks

This event is fired whenever the **Text** property has been completely checked or the spell check has been cancelled. This event is also fired after a spelling check of a single word is caused by setting the **CheckWord** property.

This event is fired even though the **BadWord** event may have been fired during a multi-word or single word spell check. Possible values for the *Cancelled* parameter are:

Value	Description
0	The spell check completed normally without being cancelled.
1	The spell check was cancelled.

CustomDictFile Property

Sets or returns the path and filename of the custom dictionary file.

Syntax

[form!]VSSpell.**CustomDictFile**[= *value* As String]

Remarks

You may have up to five custom dictionary files active in addition to the main dictionary. The custom dictionary files are specified using the **CustomDictFile**, and **CustomDictFile2** through **CustomDictFile5** properties.

Data Type

String

CustomDictFile2 Property

Sets or returns the path and filename of an additional custom dictionary file.

Syntax

[form!]VSSpell.**CustomDictFile2**[= *value* As String]

Data Type

String

CustomDictFile3 Property

Sets or returns the path and filename of an additional custom dictionary file.

Syntax

[form!]VSSpell.**CustomDictFile3**[= value As String]

Data Type

String

CustomDictFile4 Property

Sets or returns the path and filename of an additional custom dictionary file.

Syntax

[form!]VSSpell.**CustomDictFile4**[= value As String]

Data Type

String

CustomDictFile5 Property

Sets or returns the path and filename of an additional custom dictionary file.

Syntax

[form!]VSSpell.**CustomDictFile5**[= value As String]

Data Type

String

CustomDictListVisible Property

Sets or returns whether the custom dictionary list is visible on the bad word dialog box.

Syntax

Property **CustomDictListVisible** As Boolean

Data Type

Boolean

DialogAction Event

Fired when a button is clicked on the bad-word dialog box.

Syntax

Private Sub VSSpell_**DialogAction**(Action As Integer)

Remarks

The *Action* parameter identifies the button that was pressed by the user on the bad-word dialog box. Possible values for the *Action* parameter are:

Value	Constant	Description
1	DIALOGACTION_ CHANGEALL	Indicates the user pressed the Change All button on the bad-word dialog box.
2	DIALOGACTION_ CHANGE	Indicates the user pressed the Change button on the bad-word dialog box.
3	DIALOGACTION_ IGNORE	Indicates the user pressed the Ignore button on the bad-word dialog box.
4	DIALOGACTION_ IGNOREALL	Indicates the user pressed the Ignore All button on the bad-word dialog box.
5	DIALOGACTION_ ADD	Indicates the user pressed the Add button on the bad-word dialog box.
6	DIALOGACTION_ CANCEL	Indicates the user pressed the Cancel button on the bad-word dialog box.

If you set the *Action* parameter to zero while handling this event, no action will be performed. If you set the *Action* parameter to *DIALOGACTION_CANCEL*, the bad-word dialog will be dismissed and the spell-checking process will be cancelled.

Note: The bad-word dialog is displayed only if the **BadWordDialog** property is set to a non-zero value.

DialogChangedWord Property

Returns the last word typed by the user into the bad-word dialog box.

Syntax

[form!]VSSpell.**DialogChangedWord**[= *value* As String]

Data Type

String

DialogFont Property (VSSpell)

Sets or returns the font used in the bad-word dialog box.

Syntax

[form!]VSSpell.**DialogFont**[= Font]

Remarks

This property allows you to make the appearance of the built-in bad-word dialog consistent with the appearance of the main application.

Data Type

Font

Default Value

Ambient Font

DialogLeft Property (VSSpell)

Sets or returns the position (in twips) of the left edge of the bad-word dialog box relative to the left of the desktop.

Syntax

[form!]VSSpell.**DialogLeft**[= *value* As Long]

Remarks

If you set the **DialogLeft** and **DialogTop** properties to zero, the dialog will be centered on the screen. If the user moves the dialog, the new position will be reflected in these properties.

Data Type

Long

Default Value

0

DialogTitle Property (VSSpell)

Sets or returns the title displayed as the caption of the bad-word dialog box.

Syntax

[form!]VSSpell.**DialogTitle**[= *value* As String]

Data Type

String

DialogTop Property (VSSpell)

Sets or returns the position (in twips) of the top edge of the bad-word dialog box, relative to the top of the desktop.

Syntax

[form!]VSSpell.**DialogTop**[= value As Long]

Remarks

If you set the **DialogLeft** and **DialogTop** properties to zero, the dialog will be centered on the screen. If the user moves the dialog, the new position will be reflected in these properties.

Data Type

Long

Default Value

0

DictionaryIsProtected Property

Returns whether the main dictionary file was protected when it was created.

Syntax

val% = [form!]VSSpell.**DictionaryIsProtected**

Remarks

A protected dictionary cannot be dumped by the dictionary maintenance utility DICTUTIL.EXE.

Data Type

Boolean

DictionaryLanguage Property

Returns the language of the current main dictionary file.

Syntax

val& = [form!]VSSpell.**DictionaryLanguage**

Remarks

This value is selected and stored in the main dictionary file when the dictionary file is created with the dictionary maintenance utility DICTUTIL.EXE.

Possible values for this property are:

Value	**Constant**	**Description**
0	LANG_ENGLISH	English
1	LANG_SPANISH	Spanish
2	LANG_GERMAN	German
3	LANG_FRENCH	French
4	LANG_ITALIAN	Italian
5	LANG_DUTCH	Dutch
6	LANG_SWEDISH	Swedish
7	LANG_NORWAY	Norwegian
8	LANG_ICELAND	Icelandic

Data Type

Long

DictionaryVersion Property

Returns the version number of the main dictionary file.

Syntax

val& = [form!]VSSpell.**DictionaryVersion**

Remarks

A dictionary created with the dictionary utility that shipped with VSSpell 1.1 and earlier returns 1.

Later versions up to and including VSSpell 8 return 2. Version 2 dictionaries are compressed and may be protected.

Data Type

Long

DontCorrectText Property

Sets or returns whether **VSSpell** should suppress changes to the **Text** property when the user presses the Change button on the built-in bad-word dialog.

Syntax

[form!]VSSpell.**DontCorrectText**[= {True | False}]

Remarks

Setting this property to TRUE allows a read-only style of spell checking without letting the user modify the underlying text.

Data Type

Boolean

Default Value

False

HelpBtnClick Event (VSSpell)

Fired when the Help button on the bad-word dialog box is clicked.

Syntax

Private Sub VSSpell_**HelpBtnClick**()

Remarks

The **Help** button is visible only if the **HelpBtnVisible** property is set to TRUE.

HelpBtnVisible Property (VSSpell)

Sets or returns whether the Help button will be visible on the bad-word dialog box.

Syntax

[form!]VSSpell.**HelpBtnVisible**[= {True | False}]

Remarks

The Help button may be used in conjunction with the **HelpBtnClick** event to provide additional information to the user at runtime.

Data Type

Boolean

Default Value

False

IgnoreAll Property

Returns a specific word from the IgnoreAll property array.

Syntax

val$ = [form!]VSSpell.**IgnoreAll**(Index As Long)

Remarks

The **IgnoreAll** property array is zero-based. Valid indices range from zero to **IgnoreAllCount** - 1.

Elements are added to the **IgnoreAll** array when the user clicks the **Ignore All** button on the built-in bad-word dialog or by assigning a word to the **CheckWord** and setting the **IgnoreAllWord** property to TRUE.

Data Type

String

IgnoreAllCount Property

Returns the number of elements in the **IgnoreAll** string array.

Syntax

[form!]VSSpell.**IgnoreAllCount**[= value As Long]

Data Type

Long

IgnoreAllWord Property

Setting this property to TRUE adds the word in the **CheckWord** property to the IgnoreAll property array.

Syntax

[form!]VSSpell.**IgnoreAllWord** = {True | False}

Data Type

Boolean

IgnoreInMixedCase Property

Sets or returns if words with uppercase characters after lowercase characters are ignored (e.g. 'IOleObject').

Syntax

[form!]VSSpell.**IgnoreInMixedCase**[= {True | False}]

Remarks

Setting this property to TRUE makes it easy to check text that contains code samples.

Data Type

Boolean

Default Value

False

IgnoreInUpperCase Property

Sets or returns if words in all uppercase are ignored (e.g. 'RTF').

Syntax

[form!]VSSpell.**IgnoreInUpperCase**[= {True | False}]

Data Type

Boolean

Default Value

False

IgnoreWithNumbers Property

Sets or returns if words containing numbers are ignored.

Syntax

[form!]VSSpell.**IgnoreWithNumbers**[= {True | False}]

Data Type

Boolean

Default Value

False

IgnoreWord Property

Setting this property to TRUE causes the current word being checked to be ignored (this time only).

Syntax

[form!]VSSpell.**IgnoreWord** = {True | False}

Remarks

If checking multiple words with the **Text** property, spell checking continues with the next word.

Setting this property is not necessary when checking a single word using the **CheckWord** property.

To always ignore the current word, use the **IgnoreAllWord** property instead.

Data Type

Boolean

IntegerTag Property

This property is similar to the Tag standard property.

Syntax

[form!]VSSpell.**IntegerTag**[= value As Long]

Remarks

This property may be used to store any integer value and is not acted upon by **VSSpell**.

Data Type

Long

Default Value

0

LastSpellError Property

Returns the last error code that occurred on the **VSSpell** control.

Syntax

[form!]VSSpell.**LastSpellError**[= value As Long]

Remarks

Possible values for this property are:

Value	**Constant**	**Description**
0	ERR_NONE	No errors.
32001	ERR_MAINDICTFILENOTFOUND	VSSpell could not find the main dictionary file.
32002	ERR_ CANNOTOPENMAINDICTFILE	VSSpell could not open the main dictionary file.
32003	ERR_ CANNOTCREATECUSTOMDICT	VSSpell could not create the custom dictionary file.
32004	ERR_ CUSTOMDICTTOOLARGE	The custom dictionary file is too large (over 64k bytes).
32005	ERR_ALREADY CHECKING	Cannot assign text to the Text property while spell checking it.
32006	ERR_ WRONGDICTIONARYVERSION	The main dictionary file has the wrong version (VSSpell6 understands versions 1 and 2 only).

Value	**Constant**	**Description**
32007	ERR_INVALIDMAINDICTFILE	The main dictionary file is invalid (not a dictionary file).
32008	ERR_WORDTOOLONG	A word longer than 50 characters was assigned to the CheckWord property.
32009	ERR_INVALIDHWND	An invalid value was assigned to the AutoLinkHwnd property (not a window handle).

Data Type

Long

Default Value

0

MainDictFile Property

Sets or returns the fully qualified filename of the main dictionary file.

Syntax

[form!]VSSpell.**MainDictFile**[= *value* As String]

Remarks

This property must be set to the complete path and filename of a valid main dictionary file prior to any spell checking.

If a path and filename are not specified, the control will search for the default main dictionary filename (**VSSP_AE.DCT**) in the following places:

1. Application directory
2. Current directory
3. System directory
4. Windows directory
5. Path environment variable

Data Type

String

OptionBtnCaption Property (VSSpell)

Sets or returns the caption of the Option button on the bad-word dialog box.

Syntax

[form!]VSSpell.**OptionBtnCaption**[= *value* As String]

Remarks

The **Option** button is visible only if the **OptionBtnVisible** property is set to TRUE.

Data Type

String

OptionBtnClick Event (VSSpell)

Fired when the Option button on the bad-word dialog box is clicked.

Syntax

Private Sub VSSpell_**OptionBtnClick**()

Remarks

The **Option** button is visible only if the **OptionBtnVisible** property is set to TRUE.

You may use the **Option** button to display a custom dialog with spell-checking options.

OptionBtnVisible Property (VSSpell)

Sets or returns if the Option button is visible on the bad-word dialog box.

Syntax

[form!]VSSpell.**OptionBtnVisible**[= {True | False}]

Remarks

The **Option** button may be used in conjunction with the **OptionBtnClick** event to display a custom dialog with spell-checking options.

Data Type

Boolean

Default Value

False

SelLength Property

Sets or returns the length of the word currently being checked in the **Text** property.

Syntax

[form!]VSSpell.**SelLength**[= value As Long]

Data Type

Long

SelStart Property

Sets or returns the zero-based offset to the beginning of the word currently being checked in the **Text** property.

Syntax

[form!]VSSpell.**SelStart**[= *value* As Long]

Data Type

Long

Start Property

Starts the checking of the word(s) stored in the **Text** property.

Syntax

[form!]VSSpell.**Start**[= *value* As Long]

Remarks

Set the **Start** property to a non-zero value to start checking the string stored in the **Text** property.

You may also read the **Start** property to determine whether the **VSSpell** control is currently checking the string stored in the **Text** property. If it is, **Start** has a non-zero value.

Data Type

Long

Suggest Property

Sets or returns whether suggestions are generated for bad words.

Syntax

[form!]VSSpell.**Suggest**[= {True | False}]

Remarks

By default, this property is set to TRUE. Setting it to FALSE increases the speed of the spell-checking process.

The built-in bad-word dialog has a checkbox that the user may click to toggle the setting of this property on and off.

Data Type

Boolean

Default Value

True

Suggestion Property

Returns the suggested word specified by element during a **BadWord** event.

Syntax

val$ = [form!]VSSpell.**Suggestion**(*Index* As Long)

Remarks

The **Suggestion** property array is zero-based. Valid indices range from zero to **SuggestionCount** - 1.

Data Type

String

Suggestion Event (VSSpell)

Fired after a word has been added to the **Suggestion** property array.

Syntax

Private Sub VSSpell_**Suggestion**(*Problem* As Integer, *Cancel* As Integer)

Remarks

This event is fired only if the **Suggest** property is set to TRUE and there are suggestions found for the bad word. The **BadWord** event is fired after the final **Suggestion** event is fired or after a **Suggestion** event sets the *Cancel* parameter to TRUE.

The *Problem* parameter indicates the type of problem. Currently, the only problem supported by the **Suggestion** event is *NOTINDICT* (1).

Setting the *Cancel* parameter to TRUE stops the suggestion generation process. The **BadWord** event is still fired following a cancelled **Suggestion** event.

SuggestionCount Property

Returns the number of elements in the **Suggestion** string array.

Syntax

val& = [form!]VSSpell.**SuggestionCount**

Data Type

Long

Text Property

Sets or returns the string of words to be spell checked.

Syntax

[form!]VSSpell.**Text**[= *value* As String]

Remarks

You may spell check a string by assigning it to the **Text** property and then setting the **Start** property to TRUE.

Alternatively, you may assign the **AutoLinkHwnd** property to a valid window handle, and then set the **Start** property to TRUE. In this case, the **VSSpell** control will automatically retrieve the contents of the target control and will place it in the **Text** property.

During the spell-checking process, the **VSSpell** control sets the **SelStart** and **SelLength** properties to the position and length of the word being checked within the **Text** string.

Data Type

String

TypingError Event

Fired when a typing error is detected after the CheckTyping method is invoked.

Syntax

Private Sub VSSpell_**TypingError**(ByVal *SelStart* As Long, ByVal *SelLength* As Long, *Cancel* As Integer)

Remarks

The **TypingError** event indicates that the user typed a bad word. The offending word may be retrieved using the **CheckWord** property. The position and length of the offending word are indicated in the *SelStart* and *SelLength* parameters.

After the event is handled, **VSSpell** will perform the action specified by the **TypingErrorAction** property by default. If you set the *Cancel* parameter to TRUE, **VSSpell** does not perform any actions. This mechanism allows you to inform the user of typing mistakes in custom ways.

Note: This event is only fired in As-you-type Spell-Checking mode, invoked with the **CheckTyping** method. In this mode, the **Text** property is not used by **VSSpell**. This is done for efficiency since the contents of the control change constantly as the user types.

TypingErrorAction Property

Sets or returns the action to be taken when a typing error is detected.

Syntax

[form!]VSSpell.**TypingErrorAction**[= TypingErrorActionConstants]

Remarks

Typing errors are detected after the **CheckTyping** method is invoked, every time the user finishes typing a word that cannot be found in the dictionary. This property allows you to specify an action to be performed whenever a bad word is detected.

The settings for the **TypingErrorAction** property are described below:

Value	**Constant**	**Description**
0	vsspellNoAction	No action will be performed when a typing error is detected.
1	vsspellBeep	Sound a beep when a typing error is detected.
2	vsspellUnderline	Underline the offending word when a typing error is detected.

Value	Constant	Description
3	vsspellBeepAndUnderline	Sound a beep and underline the offending word when a typing error is detected.

The underline drawn to indicate typing mistakes is temporary. It is automatically erased when the user types a word that is correct or when the control is redrawn. The type of underline drawn is determined using the **UnderlineColor** and **UnderlineStyle** properties.

Data Type

TypingErrorActionConstants (Enumeration)

Default Value

vsspellBeepAndUnderline (3)

TypingOK Event

Fired when a correct word is typed after the **CheckTyping** method is invoked.

Syntax

Private Sub VSSpell_**TypingOK**(ByVal *SelStart* As Long, ByVal *SelLength* As Long)

Remarks

This event is useful in conjunction with the **TypingError** event.

Typically, you will trap the **TypingError** event to display a user-interface element (such as a message) to indicate an error. You would then trap the **TypingOK** event to hide the element.

UnderlineColor Property

Sets or returns the color of the underline effect used to highlight typing errors.

Syntax

[form!]VSSpell.**UnderlineColor**[= colorref&]

Data Type

Color

Default Value

Red (255)

UnderlineStyle Property

Sets or returns the style of the underline effect used to highlight typing errors.

Syntax

[form!]VSSpell.**UnderlineStyle**[= UnderlineStyleConstants]

Remarks

The settings for the **UnderlineStyle** property are described below:

Value	**Constant**	**Description**
0	vsspellULNone	Misspelled words are not underlined.
1	vsspellULInvert	Misspelled words are inverted.
2	vsspellULSingle	Misspelled words are underlined with a single line.
3	vsspellULDouble	Misspelled words are underlined with a double line.
4	vsspellULWiggly	Misspelled words are underlined with a wiggly line.

The color of the underline is determined by the **UnderlineColor** property.

Data Type

UnderlineStyleConstants (Enumeration)

Default Value

vsspellULWiggly (4)

WhichCustomDict Property

Sets or returns which custom dictionary to use when adding words to the custom dictionary.

Syntax

[form!]VSSpell.**WhichCustomDict**[= value As Long]

Remarks

The default value for this property is 1, which refers to **CustomDictFile** property.

Valid settings are 1 through 5, referencing the **CustomDictFile** and **CustomDictFile2** through **CustomDictFile5** properties.

Data Type

Long

Default Value

1

WordCount Property

Returns the number of individual words in the **Text** property following a spell check.

Syntax

val& = [form!]VSSpell.**WordCount**

Data Type

Long

WordsPerMinute Property

Returns the number of words processed per minute during a spell check.

Syntax

val& = [form!]VSSpell.**WordsPerMinute**

Data Type

Long

VSTHES8Lib Library

Description:	ComponentOne VSThesaurus 8.0 Control
File Name:	THES8.OCX
Help File:	VSSPELL8.CHM
GUID:	{83dcdf03-433e-11d3-bd95-c5f237c8b472}
Control:	VSThesaurus

VSThesaurus Control

Before you can use a **VSThesaurus** control in your application, you must add the **THES8.OCX** file to your project. If you use the control in most of your VB projects, you may want to add it to Visual Basic's Autoload file.

To distribute applications you create with the **VSThesaurus** control, you must install and register it on the user's computer. The Setup Wizard provided with Visual Basic provides tools to help you do this. Please refer to the Visual Basic manual for details.

VSThesaurus Properties, Events, and Methods

All of the properties, events, and methods for the **VSThesaurus** control are listed in the following tables. Properties, events, and methods that apply *only* to this control, or that require special consideration when used with it, are marked with an asterisk (*). These are documented in later sections. For documentation on the remaining properties, see the Visual Basic documentation.

Properties

***AutomaticDialog**	***ChangedWord**	***CheckWord**
DialogFont*	***DialogLeft**	***DialogTitle**
***DialogTop**	***HelpBtnVisible**	***LastThesError**
***MainThesFile**	***OptionBtnCaption**	***OptionBtnVisible**
***Synonym**	***SynonymCount**	***ThesaurusLanguage**

Events

***BadWord**	***Changed**	***Complete**
***HelpBtnClick**	***OptionBtnClick**	***Suggestion**

AutomaticDialog Property

Sets or returns whether the synonym dialog box will be used and the language that should be used to display it.

Syntax

[form!]VSThesaurus.**AutomaticDialog**[= AutomaticDialogConstants]

Remarks

Valid settings for the **AutomaticDialog** property are:

Value	**Constant**	**Description**
0	vsthesNoDialog	No synonym dialog will be displayed by the VSThes control. The calling application may choose to display its own custom dialog.
1	vsthesEnglishDialog	The English synonym dialog will be displayed.
2	vsthesSpanishDialog	The Spanish synonym dialog will be displayed.
3	vsthesGermanDialog	The German synonym dialog will be displayed.
4	vsthesFrenchDialog	The French synonym dialog will be displayed.

Data Type

AutomaticDialogConstants (Enumeration)

Default Value

vsthesEnglishDialog (1)

BadWord Event (VSThesaurus)

Fired when the word being checked is not found in the main thesaurus file.

Syntax

Private Sub VSThesaurus_**BadWord**(*Problem* As Integer)

Remarks

The problem parameter indicates the type of problem that caused the word to be bad. Currently, *NOT_IN_THES* is the only problem supported by the bad-word event.

Changed Event (VSThesaurus)

Fired to indicate that the user pressed the Replace button on the synonym dialog box.

Syntax

Private Sub VSThesaurus_**Changed**()

Remarks

You may retrieve the selected synonym by reading the **ChangedWord** property.

ChangedWord Property

Returns the synonym chosen as a replacement for the **CheckWord** property following a thesaurus check.

Syntax

val$ = [form!]VSThesaurus.**ChangedWord**

Data Type

String

CheckWord Property (VSThesaurus)

Sets a word to be looked up in the thesaurus.

Syntax

[form!]VSThesaurus.**CheckWord**[= *value* As String]

Remarks

If this property is set to a string containing more than one word, only the first word will be used.

If synonyms are found in the main thesaurus file and the **AutomaticDialog** property is set to a non-zero value, the VSThesaurus control will display its built-in synonym dialog box so the user can select a synonym to replace the word. The word selected by the user is exposed through the **ChangedWord** property. If the **AutomaticDialog** property is set to zero, no dialog will be displayed. In this case, the list of synonyms is exposed through the **Synonym** and **SynonymCount** properties.

Data Type

String

Complete Event (VSThesaurus)

Fired after the thesaurus check is performed.

Syntax

Private Sub VSThesaurus_**Complete**(Cancelled As Integer)

Remarks

The *Cancelled* parameter indicates how the thesaurus check ended. It is set to FALSE if the process ended normally, or TRUE if the process was aborted by the user.

DialogLeft Property (VSThesaurus)

Sets or returns the position (in twips) of the left edge of the synonym dialog box relative to the left of the screen.

Syntax

[form!]VSThesaurus.**DialogLeft**[= value As Long]

Remarks

If you set the **DialogLeft** and **DialogTop** properties to zero, the dialog will be centered on the screen. If the user moves the dialog, the new position will be reflected in these properties.

Data Type

Long

DialogTitle Property (VSThesaurus)

Sets or returns the title displayed as the caption of the synonym dialog box.

Syntax

[form!]VSThesaurus.**DialogTitle**[= value As String]

Data Type

String

DialogTop Property (VSThesaurus)

Sets or returns the position (in twips) of the top edge of the synonym dialog box relative to the top of the screen.

Syntax

[form!]VSThesaurus.**DialogTop**[= value As Long]

Remarks

If you set the **DialogLeft** and **DialogTop** properties to zero, the dialog will be centered on the screen. If the user moves the dialog, the new position will be reflected in these properties.

Data Type

Long

HelpBtnClick Event (VSThesaurus)

Fired when the Help button on the synonym dialog box is clicked.

Syntax

Private Sub VSThesaurus_**HelpBtnClick**()

Remarks

The **Help** button is visible only if the **HelpBtnVisible** property is set to TRUE.

HelpBtnVisible Property (VSThesaurus)

Sets or returns whether the Help button will be visible on the synonym dialog box.

Syntax

[form!]VSThesaurus.**HelpBtnVisible**[= {True | False}]

Remarks

The Help button may be used in conjunction with the **HelpBtnClick** event to provide additional information to the user at runtime.

Data Type

Boolean

LastThesError Property

Returns the code of the last error that occurred on the control.

Syntax

[form!]VSThesaurus.**LastThesError**[= value As Long]

Data Type

Long

MainThesFile Property

Sets or returns the fully qualified filename of the main thesaurus file.

Syntax

[form!]VSThesaurus.**MainThesFile**[= value As String]

Remarks

This property must be set to the complete path and filename of a valid thesaurus file prior to any spell checking.

If a path and filename are not specified, the control will search for the default thesaurus filename (**VSSP_AE.THE**) in the following places:

1. Application directory
2. Current directory
3. System directory
4. Windows directory
5. Path environment variable

Data Type

String

OptionBtnCaption Property (VSThesaurus)

Sets or returns the caption of the Option button on the synonym dialog box.

Syntax

[form!]VSThesaurus.**OptionBtnCaption**[= value As String]

Remarks

The **Option** button is visible only if the **OptionBtnVisible** property is set to True.

Data Type

String

OptionBtnClick Event (VSThesaurus)

Fired when the Options button on the synonym dialog box is clicked.

Syntax

Private Sub VSThesaurus_**OptionBtnClick**()

Remarks

The **Option** button is visible only if the **OptionBtnVisible** property is set to TRUE.

OptionBtnVisible Property (VSThesaurus)

Sets or returns if the Option button is visible on the synonym dialog box.

Syntax

[form!]VSThesaurus.**OptionBtnVisible**[= {True | False}]

Data Type

Boolean

Suggestion Event (VSThesaurus)

Fired after each synonym is added to the **Synonym** property array.

Syntax

Private Sub VSThesaurus_**Suggestion**(*Problem* As Integer, *Cancel* As Integer)

Remarks

This event is fired once for each synonym found for the word being checked. The *Problem* parameter is always set to zero.

Set the *Cancel* parameter to TRUE to stop suggesting synonyms.

Synonym Property

Returns a synonym found for the word in the **CheckWord** property.

Syntax

val$ = [form!]VSThesaurus.**Synonym**(*Index* As Long)

Remarks

The **Synonym** property array is zero-based. Valid indices range from zero to **SynonymCount** - 1.

The **Synonym** array is built by the VSThesaurus control every time a word is assigned to the **CheckWord** property.

Data Type

String

SynonymCount Property

Returns the number of elements in the **Synonym** string array.

Syntax

val& = [form!]VSThesaurus.**SynonymCount**

Data Type

Long

ThesaurusLanguage Property

Returns the language of the words in the current main thesaurus file.

Syntax

val& = [form!]VSThesaurus.**ThesaurusLanguage**

Data Type

Long

VSSpell Control Error Codes

The following is a list of **VSSpell** error codes that can be trapped. We recommend that you declare these values as either global of local constants and compare any error codes to this list of values. You can change the names of these constants, but we suggest that you choose something that is descriptive of the error that occurred.

Value	**Constant**	**Description**
0	ERR_NONE	No errors.
32001	ERR_ MAINDICTFILENOTFOUND	**VSSpell** could not find the main dictionary file.
32002	ERR_ CANNOTOPENMAINDICTFIL	**VSSpell** could not open the main dictionary file.
32003	ERR_ CANNOTCREATECUSTOMDICT	**VSSpell** could not create the custom dictionary file.
32004	ERR_ CUSTOMDICTTOOLARGE	The custom dictionary file is too large (over 64k bytes).
32005	ERR_ ALREADYCHECKING	Cannot assign text to the **Text** property while spell checking it.
32006	ERR_ WRONGDICTIONARYVERSION	The main dictionary file has the wrong version (VSSpell6 understands versions 1 and 2 only).
32007	ERR_ INVALIDMAINDICTFILE	The main dictionary file is invalid (not a dictionary file).

Value	**Constant**	**Description**
32008	ERR_ WORDTOOLONG	A word longer than 50 characters was assigned to the **CheckWord** property.
32009	ERR_ INVALIDHWND	An invalid value was assigned to the **AutoLinkHwnd** property (not a window handle).

VSThesaurus Control Error Codes

The following is a list of **VSThesaurus** error codes that can be trapped. We recommend that you declare these values as either global of local constants and compare any error codes to this list of values. You can change the names of these constants, but we suggest that you choose something that is descriptive of the error that occurred.

Value	Constant	Description
32001	MAIN_THES_FILE_ NOT_FOUND	The specified in the **MainThesFile** property could not be found. Specify a valid main thesaurus file. This error can occur when you start a thesaurus check with the **Start** property or the **CheckWord** property.
32002	CANNOT_OPEN_ MAIN_THES_FILE	The main thesaurus file could not be opened. Makes sure that there are enough file handles available and the file is not locked by another process.

Index