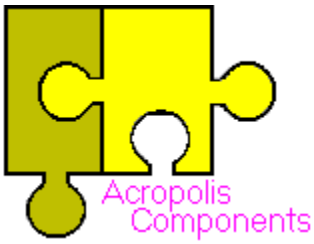

Acropolis Spell Checking Components

The Acropolis Spell Checking Components are a group of components designed to allow developers to quickly and easily add spell checking to their Delphi applications.

It consists of the following three components:

AcropSpell (Both 16 & 32-Bit versions)

MemoSpell (Both 16 & 32-Bit versions)

OrphSpell (16-Bit Delphi 1.0 component)

RichSpell (32-Bit Delphi 2.0 component)

You can also use the Base Spelling Unit, but it is not a component and should only be used by programmers familiar with Object Pascal and creating components.

AcropSpell Component

Properties Methods

Units
AcropSpl & AcropS32

Description
The AcropSpell component is the main component in the group of Acropolis spell checking components and allows for low-level access to the spell checking engine found in the BaseASpl & BsASpl32 units. It is the lowest level component in the group and requires you to write all the code to break the text into individual words, present the user with the list of suggestions and handle replacement of the misspelled words with the corrected word. Essentially all of the user interface of the spell checker. The typical code to use the unit will follow steps something like the following:

1. Clear any ignore or replace word lists (if you allow them).
2. Start at the beginning of the text to spell check.
3. Get the next word in the text to check.
4. If the word isn't in your ignore/replace list use the GoodWord method to see if it is in the dictionaries.
5. If the word isn't in the dictionaries use the SuggestCloseMatch or SuggestPhoneme methods to create a list of suggestions and then allow them to select a word from the list, edit it or indicate that they want to ignore or replace all occurrences of the word.
6. Perform the action indicated in step 5.
7. Return to step 3 until all the text has been checked.

Additionally, you will have to Open and Close the dictionaries (both the main and user) as appropriate.

AcropSpell Properties

DictionaryName
DictionaryUser
MaxSuggestions

DictionaryName Property

Applies to
AcropSpell component

This property sets the name of the main dictionary. It must include the full path of the dictionary file.
The default value of this property is **ACROP.DCT**.
The best place to set this property is when you create your form or where you are reading in the INI file for your applications (if they differ). If you place the **ACROP.DCT** file in the subdirectory with your application you can leave the value as it is.

Example: AcropSpl1.DictionaryName := 'C:\MYAPP\ACROP.DCT';

See Also: DictionaryUser

DictionaryUser Property

Applies to
AcropSpell component

This property sets the name of the default user dictionary. It must include the full path of the dictionary file.
The default value of this property is **CUSTOM.DCT**.
The best place to set this property is when you create your form or where you are reading in the INI file for your applications (if they differ). If you place the **CUSTOM.DCT** file in the subdirectory with your application (where it will be created if you haven't supplied a full path name) you can leave the value as it is.

Example: AcropSpl1.DictionaryUser := 'C:\MYAPP\CUSTOM.DCT';

See Also: DictionaryName

MaxSuggestions Property

Applies to
AcropSpell, MemoSpell, OrphSpell &RichSpell

This property sets the maximum number of suggestions returned for the MemoSpell, MemoSpell and OrphSpell components. With the AcropSpell component it controls the number of suggestions returned by the SuggestCloseMatch and SuggestPhoneme methods.

The default value of this property is 10. The smallest this value can be is 1 and the largest is 255.

The more suggestions you ask for the slower the two suggestion methods will be. A good typical value for MaxSuggestions is 10-15. After the 10th suggestion the words returned start having little relationship to the actual word.

See Also: In relation to the AcropSpell component:
SetMaxSuggestions
SuggestCloseMatch
SuggestPhoneme

Example: AcropSpl1.MaxSuggestions := 15;

AcropSpell Methods

AddWord
BuildUserDictionary
CloseDictionaries
CloseUserDictionaries
CloseUserDictionary
DeleteUserDictionaries
DeleteUserDictionary
GetUserDictionaryList
GoodWord
IsDictionarieyOpen
OpenDictionary
OpenUserDictionary
SetMaxSuggestions
SuggestCloseMatch
SuggestPhoneme

AddWord Method

Applies to
AcropSpell component

Declaration
function AddWord(TheWord : **string**; DictID : integer) : boolean;

Description
The AddWord method is used to add a word on a user dictionary. The word to add is in the TheWord parameter and the DictID parameter indicated which currently opened user dictionaries to add the word to. If the word is already in any open dictionary (either the main or any of the user dictionaries) it will not be added to the indicated dictionary. AddWord will return TRUE if the word was successfully added to the dictionary. It will return FALSE if the word was not added to the dictionary. Reasons for the word not being added to the dictionary include: An invalid DictID was pasted to it, the word was already in on of the open dictionaries, or there was not available disk space expand the dictionary file.

If the characters in the word passed to the GoodWord method are not part of the allowed Character Set they will be stripped from the word before it is added to the dictionary.

Example:
```
if not AcropSpell1.AddWord(CurrentWord) then
  Form1.AddWordError;
```

BuildUserDictionary Method

Applies to
AcropSpell component

Declaration
function BuildUserDictionary(Filename : **string**; WordList : TStringList) : integer;

Description
The BuildUserDictionary method will create a user dictionary from scratch using the list of words you pass to it in the WordList parameter. If there is an existing dictionary using the requested filename it will be deleted before new dictionary is created. The newly created dictionary will also be opened ready for use. The value returned by BuildUserDictionary is the DictID used in the other user dictionary methods such as: AddWord, CloseUserDictionaries, DeleteUserDictionary, and GetUserDictionaryList.

You must include the full path as part of the dictionary's filename.

If the dictionary could not be created the value returned will be -1.

The BuildUserDictionary method is also used in Deleting Words From The Dictionary.

See Also: AddWord
CloseUserDictionaries
DeleteUserDictionary
GetUserDictionaryList
OpenUserDictionary

Example:
```
EmptyList.Clear;
NewID := BuildUserDictionary('CUSTOM.DCT', EmptyList);
if NewID = -1 then
  CannotOpenDictionaryError
else
  Form1.UserDictID := NewID;
```

CloseDictionaries Method

Applies to
AcropSpell component

Declaration
procedure CloseDictionaries;

Description
The CloseDictionaries method will close all open dictionaries, including the main dictionary and all open user dictionaries.

See Also: CloseUserDictionary
CloseUserDictionaries

CloseUserDictionaries Method

Applies to
AcropSpell component

Declaration
procedure CloseUserDictionaries;

Description
The CloseUserDictionaries method will close all open user dictionaries.

See Also: CloseUserDictionary
CloseDictionaries

CloseUserDictionary Method

Applies to
AcropSpell component

Declaration
function CloseUserDictionary(DictID : integer) : boolean;

Description
The CloseUserDictionary method will close the user dictionary identified with the DictID value you pass to it. If the dictionary was closed successfully it will return TRUE. If the dictionary could not be close (such as you passed it an invalid DictID value) it will return FALSE.

See Also: CloseUserDictionaries
CloseDictionaries

DeleteUserDictionaries Method

Applies to
AcropSpell component

Declaration
procedure DeleteUserDictionaries;

Description
The DeleteUserDictionaries method should be used with ***extreme caution*** as it will close and ***physically delete*** all the currently open user dictionary files. There is really little use for this method unless your component is designed to deal with only one user dictionary at a time. It is far better and less dangerous to use the DeleteUserDictionary method. The primary use for these two methods are in Deleting Words From The Dictionary.

See Also: DeleteUserDictionary

DeleteUserDictionary Method

Applies to
AcropSpell component

Declaration
function DeleteUserDictionary(DictID : integer) : boolean;

Description
The DeleteUserDictionary method should be used with ***caution*** as it will close and ***physically delete*** the specified user dictionary file. The primary use for this method is in Deleting Words From The Dictionary. DeleteUserDictionary will return TRUE if the dictionary file was properly closed and deleted. It will return FALSE if the file could not be closed or deleted or if you passed an invalid DictID value to it.

See Also: DeleteUserDictionaries

GetUserDictionaryList Method

Applies to
AcropSpell component

Declaration
function GetUserDictionaryList(DictID : integer) : TStringList;

Description
The GetUserDictionary method will return a sorted TStringList containing all of the words in the specified user dictionary. The primary use for this method is in Deleting Words From The Dictionary.

Example: DictionaryForm.WordListBox.Items := GetUserDictionaryList(Form1.UserDictID);

GoodWord Method

Applies to
AcropSpell component

Declaration
function GoodWord(TheWord : **string**) : boolean;

Description
The GoodWord method will return TRUE if the word in TheWord was found in either the main dictionary or any of the open user dictionaries. It will return FALSE if the word was not found in any of the open dictionaries or if there are currently no open dictionaries.

If there are characters in the word that are not part of the allowed Character Set then they be stripped from word before it is tested.

Example:
```
if not AcropSpell1.GoodWord(CurrentWord) then
    Form1.SuggestReplacements(CurrentWord);
```

IsDictionaryOpen Method

Applies to
AcropSpell component

Declaration
function IsDictionaryOpen : boolean;

Description
The IsDictionaryOpen method will return TRUE if the main dictionary is open. Otherwise it will return FALSE.

Example:
```
if not AcropSpell1.IsDictionaryOpen then
    AcropSpell1.OpenDictionary(Form1.MainDictionaryName);
```

OpenDictionary Method

Applies to
AcropSpell component

Declaration
function OpenDictionary(Filename : **string**) : boolean;

Description
The OpenDictionary method is used to open the main dictionary. You must include the full path as part of the filename. Only one main dictionary may be opened at a time. If the dictionary was opened successfully OpenDictionary will return TRUE. If the dictionary file could not be opened, is not a primary dictionary file (i.e. a user dictionary) or there is already a main dictionary open FALSE will be returned.

See Also: OpenUserDictionary

Example:
```
if not AcropSpell1.OpenDictionary(Form1.MainDictionaryName) then
  Form1.OpeningDictionaryError;
```

OpenUserDictionary Method

Applies to
AcropSpell component

Declaration
function OpenUserDictionary(Filename : **string**) : integer;

Description
The OpenUserDictionary method is used to open an existing user dictionary. You must include the full path as part of the filename passed to it. If the dictionary was opened successfully, the value returned is the DictID for that dictionary and will be a positive number. It is important to remember the DictID returned as it is required to use the opened user dictionary with the AddWord, CloseUserDictionary, DeleteUserDictionary, and GetUserDictionaryList methods.

If the dictionary could not be opened (i.e. the file was not found or it was not a user dictionary file) -1 will be returned.

See Also: AddWord
CloseUserDictionary
DeleteUserDictionary
GetUserDictionaryList

Example:
```
NewID := AcropSpell1.OpenUserDictionary(Form1.UserDictionaryName);
if NewID <> -1 then
  Form1.UserDictID := NewID
else
  Form1.OpenUserDictionaryError;
```

SetMaxSuggestions Method

Applies to
AcropSpell component

Declaration
procedure SetMaxSuggestions(Max : byte);

Description
The SetMaxSuggestions method is the method call equivalent of the MaxSuggestions property. Passing it a value will set the number of suggestion returned by the SuggestCloseMatch and
SuggestPhoneme methods. The largest you can set the maximum number of suggestions to is 255. The default maximum number of suggestions is **10**.
The more suggestions you ask for the slower the two suggestion methods will be. A good typical value is 10-15. After the 10th suggestion the words returned start having little relationship to the actual word.

See Also: MaxSuggestions
SuggestCloseMatch
SuggestPhoneme

Example: AcropSpell1.SetMaxSuggestions(15);

SuggestCloseMatch Method

Applies to
AcropSpell component

Declaration
function SuggestCloseMatch(TheWord : **string**) : TStringList;

Description
The SuggestCloseMatch method will create a list of suggested correct spellings for the word passed in TheWord. The number of suggestions returned is controlled by the MaxSuggestions property or the SetMaxSuggestions method.

The list will be order based on how close the method believes the spelling is the correct spelling of the word you passed it. The first word in the list will be the most likely, the second the second most likely and so on.

The SuggestCloseMatch method returns suggestions based on a Close Match style of suggestions.

See Also: SuggestPhoneme
MaxSuggestions
SetMaxSuggestions

Example:
```
Form1.SuggestDialog.SugList.Items :=
    AcropSpell1.SuggestCloseMatch(CurrentWord);
```

SuggestClosePhoneme Method

Applies to
AcropSpell component

Declaration
function SuggestClosePhoneme(TheWord : **string**) : TStringList;

Description
The SuggestPhoneme method will create a list of suggested correct spellings for the word passed in TheWord. The number of suggestions returned is controlled by the MaxSuggestions property or the SetMaxSuggestions method.

The list will be order based on how close the method believes the spelling is the correct spelling of the word you passed it. The first word in the list will be the most likely, the second the second most likely and so on.

The SuggestPhoneme method returns suggestions based on a Phoneme Match style of suggestions.

See Also: SuggestCloseMatch
MaxSuggestions
SetMaxSuggestions

Example:
```
Form1.SuggestDialog.SugList.Items :=
    AcropSpell1.SuggestPhoneme(CurrentWord);
```

RichSpell Component

Properties Methods

Unit
RichCk32

Description
The RichSpell component is the component in the group of Acropolis spell checking components that allows for quick and easy spell checking of standard TRichEdit components. If you place the dictionary in the same directory as the executable of your application you do not even have to set the DictionaryMain or DictonaryUser properties. However, typically you will want to set these values when you read in the INI file information for your applications. The other important properties for the RichSpell component are the SuggestType which allows you to set the default type of suggestion the user is initially give, you can set it at design time until you want them to be able to select a default type of suggestion. The MaxSuggestions property which sets the maximum number of suggestions returned and the LeaveDictionariesOpen property which allows RichSpell to leave the dictionary files open to increase its speed.

MemoSpell Component

Properties Methods

Units
MemoChk & MemoCk32

Description
The MemoSpell component is the component in the group of Acropolis spell checking components that allows for quick and easy spell checking of standard TMemo and TDBMemo components. If you place the dictionary in the same directory as the executable of your application you do not even have to set the DictionaryMain or DictonaryUser properties. However, typically you will want to set these values when you read in the INI file information for your applications. The other important properties for the MemoSpell component are the SuggestType which allows you to set the default type of suggestion the user is initially give, you can set it at design time until you want them to be able to select a default type of suggestion. The MaxSuggestions property which sets the maximum number of suggestions returned and the LeaveDictionariesOpen property which allows MemoSpell to leave the dictionary files open to increase its speed.

RichSpell Properties

SuggestType
DictionaryMain
DictonaryUser
MaxSuggestions
LeaveDictionariesOpen
AvoidHighlight

RichSpell Methods

CheckRich
CheckRichSelection
ClearLists

MemoSpell & RichSpell Properties

SuggestType
DictionaryMain
DictonaryUser
MaxSuggestions
LeaveDictionariesOpen
AvoidHighlight

MemoSpell & RichSpell Methods

CheckMemo
CheckMemoSelection
CheckDBMemo
CheckDBMemoSelection
ClearLists

SuggestType Property

Applies to
MemoSpell, OrphSpell, & RichSpell

This property sets the default type of suggestions that will be shown to the user when the suggestion dialog box is displayed.

The types of suggestions are:

stNoSuggest	Generate no suggestion list.
stCloseMatch	Generate a list using the Close Match method.
stPhoneme	Generate a list using the Phoneme method.

Example:
```
MemoSpell1.SuggestType  := stPhoneme;
                   or
OrphSpell1.SuggestType := stPhoneme;
```

DictionaryMain Property

Applies to
MemoSpell, OrphSpell, & RichSpell

This property sets the name of the main dictionary. It must include the full path of the dictionary file.
The default value of this property is **ACROP.DCT**.
The best place to set this property is when you create your form or where you are reading in the INI file for your applications (if they differ). If you place the **ACROP.DCT** file in the subdirectory with your application you can leave the value as it is.

Example: MemoSpell1.DictionaryMain := 'C:\MYAPP\ACROP.DCT';

See Also: DictionaryUser

DictionaryUser Property

Applies to
MemoSpell, OrphSpell, & RichSpell

This property sets the name of the user dictionary. It must include the full path of the dictionary file.
The default value of this property is **CUSTOM.DCT**.

The best place to set this property is when you create your form or where you are reading in the INI file for your applications (if they differ). If you want the **CUSTOM.DCT** file in the subdirectory with your application you can leave the value as it is.

Note: With the user dictionary being a separate file you can have your application use the main dictionary file from where the application is installed on a server and then allow the user dictionary file to be placed on the user's local machine allowing each user of a networked application to have their own custom dictionary.

Example: MemoSpell1.DictionaryUser := 'C:\MYAPP\CUSTOM.DCT';

See Also: DictionaryMain

LeaveDictionariesOpen Property

Applies to
MemoSpell, OrphSpell, & RichSpell

The LeaveDictionariesOpen property allows MemoSpell and OrphSpell to leave their dictionary files open between calls to their methods. This has a major speed advantage as each time the main dictionary is opened the internal cache for it is cleared and has to be refilled. This causes the first few attempts to check if a word is in the dictionary and first one or two generations of suggestion list to take a little longer until the cache is refilled with parts of the main dictionary. The only drawback to setting LeaveDictionariesOpen to TRUE is that there will be two files (one for the main and one for the user dictionary) open at all times. If you cannot afford the file resources by all means set LeaveDictionariesOpen to FALSE. The default value is TRUE which leaves the dictionary files open.

Example: MemoSpell1.LeaveDictionariesOpen := TRUE;

AvoidHighlight Property

Applies to
MemoSpell, OrphSpell, & RichSpell

The AvoidHighlight property tells MemoSpell and OrphSpell if their dialog window should automatically attempt to avoid covering the highlighted selection in the editor control. If set to TRUE the dialog window will move only if it will be covering the highlighted selection. If it is set to FALSE no attempt will be made to avoid covering the highlighted selection.

NOTE: The AvoidHighlight is implemented for the RichSpell component, but does not function due to a bug in the TRichEdit component that cause an exception when the EM_POSFROMCHAR message is sent to it. Regardless of the setting of the AvoidHighlight with the RichSpell component the dialog box will aways appear in the center of the screen the first time it is used and where the user moved it to from then on.

Example: MemoSpell1.AvoidHighlight := TRUE;

CheckRich Method

Applies to
RichSpell component

Declaration
procedure CheckRich(TheMemo : TRichEdit);

Description
The CheckRich method is the main method of the RichSpell component.

The method accepts one parameter, the RichEdit component to be spell checked. If the component has been changed by correcting some of the words the Modified property will be set to TRUE.

See Also: CheckRichSelection

Example: RichSpell1.CheckRich(RichEdit1);

CheckRichSelection Method

Applies to
RichSpell component

Declaration
procedure CheckRichSelection(TheMemo : TRichEdit);

Description
The CheckRichSelection method is the alternate method of the RichSpell component. It works just like the main CheckRich method except it only check the spelling of the text that is currently selected in the component. If there is no text currently selected it will immediately exit.

The method accepts one parameter, the RichEdit component to be spell checked. If the component has been changed by correcting some of the words the Modified property will be set to TRUE.

See Also: CheckRich

Example: RichSpell1.CheckRichSelection(RichEdit1);

CheckMemo Method

Applies to
MemoSpell component

Declaration
procedure CheckMemo(TheMemo : TMemo);

Description
The CheckMemo method is the main method of the MemoSpell component. In fact even the CheckDBMemo method type casts the TDBMemo into a TMemo to perform the spell checking.

The method accepts one parameter, the Memo to be spell checked. If the memo has been changed by correcting some of the words the Modified property will be set to TRUE.

See Also: CheckMemoSelection
CheckDBMemo

Example: MemoSpell1.CheckMemo(Memo1);

CheckMemoSelection Method

Applies to
MemoSpell component

Declaration
procedure CheckMemoSelection(TheMemo : TMemo);

Description
The CheckMemoSelection method is the alternate method of the MemoSpell component. It works just like the main CheckMemo method except it only check the spelling of the text that is currently selected in the memo. If there is no text currently selected it will immediately exit.

The method accepts one parameter, the Memo to be spell checked. If the memo has been changed by correcting some of the words the Modified property will be set to TRUE.

See Also: CheckMemo
CheckDBMemo
CheckDBMemoSelection

Example: MemoSpell1.CheckMemoSelection(Memo1);

CheckDBMemo Method

Applies to
MemoSpell component

Declaration
procedure CheckDBMemo(TheMemo : TDBMemo);

Description
The CheckDBMemo method is other method of the MemoSpell component. It calls the CheckMemo method to perform the spell checking.

The method accepts one parameter, the DBMemo to be spell checked. If the memo has been changed by correcting some of the words the Modified property will be set to TRUE.

See Also: CheckDBMemoSelection
CheckMemo

Example: MemoSpell1.CheckDBMemoSelection(DBMemo1);

CheckDBMemoSelection Method

Applies to
MemoSpell component

Declaration
procedure CheckDBMemoSelection(TheMemo : TDBMemo);

Description
The CheckDBMemoSelection method is alternate version of the CheckDBMemo method. It works just the same except it only checks the text that is currently selected. If there is currently not selected text it will exit immediately.

The method accepts one parameter, the DBMemo to be spell checked. If the memo has been changed by correcting some of the words the Modified property will be set to TRUE.

See Also: CheckDBMemo
CheckMemo
CheckMemoSelection

Example: MemoSpell1.CheckDBMemo(DBMemo1);

ClearLists Method

Applies to
MemoSpell, OrphSpell, & RichSpell components

Declaration
procedure ClearLists;

Description
Internally the components maintains two lists. One is the list of words to ignore when spell checking and the other is a list of words to replace (and the words to replace them with) when spell checking. Both lists are automatically cleared when the component is created. However, after that the lists will be kept until the component is destroyed or freed. The ClearLists method will clear both of these lists to allow the user of your application to start with a fresh empty list of words to ignore or replace. You would typically want to clear the lists after you load new data into the memo/editor.

Example: MemoSpell1.ClearLists;

OrphSpell Component

Properties Methods

Unit
OrphChk

Description
The OrphSpell component is the component in the group of Acropolis spell checking components that allows for quick and easy spell checking of Turbo Power Software's Orpheus CustomEditor components. If you place the dictionary in the same directory as the executable of your application you do not even have to set the DictionaryMain or DictonaryUser properties. However, typically you will want to set these values when you read in the INI file information for your applications. The other important properties for the MemoSpell component are the SuggestType which allows you to set the default type of suggestion the user is initially give, you can set it at design time until you want them to be able to select a default type of suggestion. The MaxSuggestions property which sets the maximum number of suggestions returned and the LeaveDictionariesOpen property which allows OrphSpell to leave the dictionary files open to increase its speed.

IMPORTANT:
This component ***requires*** that you have Turbo Power Software's Orpheus components installed. It makes use of the Orpheus TOVCCustomEditor so you can spell check the following Orpheus editor types with the CheckOrph method: TOvcCustomEditor, TOvcEditor, TOvcCustomTextEditor, TOvcTextFileEditor and TOvcdbEditor. None of Turbo Power's units, components or source is included with the Acropolis Spell Checking Components as that would be illegal redistribution of their product. However, if you have a need for a large editor (files up to 16 megabytes) to replace TMemo I would highly recommend you purchase Turbo Power's Orpheus package. Besides the large editors it also provides a large number of useful and powerful components for Delphi such as:

Large virtual list boxes, numerous data entry types, array editors,
Table components, spinners, rotated labels, timers and much more.

Turbo Power Software can be reached at: 1-800-333-4160

You must have Orpheus installed to use this component. In fact this component will not even install unless Turbo Power's Orpheus is already installed in your copy of Delphi.

OrphSpell Properties

SuggestType
DictionaryMain
DictonaryUser
MaxSuggestions
LeaveDictionariesOpen
AvoidHighlight

OrphSpell Methods

CheckOrph
CheckOrphSelection
ClearLists

CheckOrph Method

Applies to
OrphSpell component

Declaration
procedure CheckOrph(TheEditor : TOvcCustomEditor);

Description
CheckOrph is the main method of the OrphSpell component. It allows you to spell check any of the Orpheus editors descended from TOvcCustomEditor such at TOvcEditor, TOvcCustomTextFileEditor, TOvcTextFileEditor and TOvcDbEditor.

You must have the Turbo Power Software's Orpheus package installed for the CheckOrph component to work. In fact the component will not even install unless you have Orpheus installed.

The method accepts one parameter, the Orpheus editor to be spell checked. If the editor has been changed by correcting some of the words the Modified property will be set to TRUE.

See Also: CheckOrphSelection

Example: OrphSpell1.CheckOrph(OvcTextFileEditor1);

CheckOrphSelection Method

Applies to
OrphSpell component

Declaration
procedure CheckOrphSelection(TheEditor : TOvcCustomEditor);

Description
CheckOrphSelection is the alternate method to the method. It works the same except it will only test the spelling of the currently selected text. If there is not text selected it will exit immediately.

You must have the Turbo Power Software's Orpheus package installed for the CheckOrph component to work. In fact the component will not even install unless you have Orpheus installed.

The method accepts one parameter, the Orpheus editor to be spell checked. If the editor has been changed by correcting some of the words the Modified property will be set to TRUE.

See Also: CheckOrph

Example: OrphSpell1.CheckOrphSelection(OvcTextFileEditor1);

ClearLists Method

Applies to
OrphSpell component

Declaration
procedure ClearLists;

Description
Internally the OrphSpell component maintains two lists. One is the list of words to ignore when spell checking and the other is a list of words to replace (and the words to replace them with) when spell checking. Both lists are automatically cleared when the component is created. However, after that the lists will be kept until the component is destroyed or freed. The ClearLists method will clear both of these lists to allow the user of your application to start with a fresh empty list of words to ignore or replace. You would typically want to clear the lists after the you load new data into the Orpheus editor you are spell checking.

Example: OrphSpell1.ClearLists;

Close Match Suggestions

Close Match suggestions are created based on common spelling and typing errors such as missing a key and hitting one near it or reversing the order of two characters, not pressing the spacebar and so forth.

Phoneme Suggestions

Phoneme suggestions are created based on they way a word sounds. This is a good method to use for catching spelling errors of people that spell a word the way it sounds and not necessarily the way it is actually spelled.

Acropolis Spell Base Unit

procedures/functions types

Units
BaseASpl & BsASpl32

Description
This is the base unit in the Acropolis Spell checking components. It is not a component, but a unit used by the other components to do the actual spell checking. To use this unit you should be familiar with Object Pascal programming.

The most important thing to remember in accessing this unit is that you must call the InitDictionaryData procedure and store the pointer returned by it before using the other functions and procedures and then you must call the ReleaseDictionaryData procedure when finally finished using the unit. This is to allow the unit to deal with multiple instances of the components using it. The pointer returned will point to a structure used internally by the unit to store caching information, dictionary file information and so on. All the other procedures and functions require the pointer be passed to them in order to operate properly.

Base Unit Procedures & Functions

AddWord
BuildUserDictionary
CloseDictionaries
CloseUserDictionaries
CloseUserDictionary
DeleteUserDictionaries
DeleteUserDictionary
GoodWord
GetUserDictionaryList
InitDictionaryData
OpenDictionary
OpenUserDictionary
ReleaseDictionaryData
SuggestCloseMatch
SuggestPhoneme

Base Unit Types

DictPtr
Character Set

AddWord Function

Applies to
ABaseSpl unit

Declaration
function AddWord(DictPtr: pointer; TheWord : **string**; DictID : integer) : boolean;

Description
The AddWord function is used to add a word on a user dictionary. The word to add is in the TheWord parameter and the DictID parameter indicates which currently opened user dictionaries to add the word to. If the word is already in any open dictionary (either the main or any of the user dictionaries) it will not be added to the indicated dictionary. AddWord will return TRUE if the word was successfully added to the dictionary. It will return FALSE if the word was not added to the dictionary. Reasons for the word not being added to the dictionary include: An invalid DictID was pasted to it, the word was already in on of the open dictionaries, or there was not available disk space expand the dictionary file.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

If the characters in the word passed to the GoodWord method are not part of the allowed Character Set they will be stripped from the word before it is added to the dictionary.

Example:
```
if not ABaseSpl.AddWord(MyComp.DictPtr, CurrentWord) then
  Form1.AddWordError;
```

BuildUserDictionary Function

Applies to
ABaseSpl unit

Declaration
function BuildUserDictionary(DictPtr: pointer; Filename : **string**;
WordList : TStringList) : integer;

Description
The BuildUserDictionary function will create a user dictionary from scratch using the list of words you pass to it in the WordList parameter. If there is an existing dictionary using the requested filename it will be deleted before new dictionary is created. The newly created dictionary will also be opened ready for use. The value returned by BuildUserDictionary is the DictID used in the other user dictionary procedures and functions such as: AddWord, CloseUserDictionaries, DeleteUserDictionary, and GetUserDictionary.

You must include the full path as part of the dictionary's filename.

If the dictionary could not be created the value returned will be -1.

The BuildUserDictionary method is also used in Deleting Words From The Dictionary.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

See Also: AddWord
CloseUserDictionaries
DeleteUserDictionary
GetUserDictionary
OpenUserDictionary

Example:
```
EmptyList.Clear;
NewID := ABaseSpl.BuildUserDictionary(MyComp.DictPtr, 'CUSTOM.DCT', EmptyList);
if NewID = -1 then
  CannotOpenDictionaryError
else
  MyComp.UserDictID := NewID;
```

DeleteUserDictionaries Procedure

Applies to
ABaseSpl unit

Declaration
procedure DeleteUserDictionaries(DictPtr: pointer);

Description
The DeleteUserDictionaries procedure should be used with ***extreme caution*** as it will close and ***physically delete*** all the currently open user dictionary files. There is really little use for this method unless your component is designed to deal with only one user dictionary at a time. It is far better and less dangerous to use the DeleteUserDictionary method. The primary use for these two methods are in Deleting Words From The Dictionary.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

See Also: DeleteUserDictionary

DeleteUserDictionary Function

Applies to
ABaseSpl unit

Declaration
function DeleteUserDictionary(DictPtr: pointer; DictID : integer) : boolean;

Description
The DeleteUserDictionary function should be used with ***caution*** as it will close and ***physically delete*** the specified user dictionary file. The primary use for this method is in Deleting Words From The Dictionary. DeleteUserDictionary will return TRUE if the dictionary file was properly closed and deleted. It will return FALSE if the file could not be closed or deleted or if you passed an invalid DictID value to it.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

See Also: DeleteUserDictionaries

CloseDictionaries Procedure

Applies to
ABaseSpl unit

Declaration
procedure CloseDictionaries(DictPtr: pointer);

Description
The CloseDictionaries procedure will close all open dictionaries, including the main dictionary and all open user dictionaries.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

See Also: CloseUserDictionary
CloseUserDictionaries

CloseUserDictionaries Procedure

Applies to
ABaseSpl unit

Declaration
procedure CloseUserDictionaries(DictPtr: pointer);

Description
The CloseUserDictionaries procedure will close all open user dictionaries.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

See Also: CloseUserDictionary
CloseDictionaries

CloseUserDictionary Function

Applies to
ABaseSpl unit

Declaration
function CloseUserDictionary(DictPtr: pointer; DictID : integer) : boolean;

Description
The CloseUserDictionary function will close the user dictionary identified with the DictID value you pass to it. If the dictionary was closed successfully it will return TRUE. If the dictionary could not be close (such as you passed it an invalid DictID value) it will return FALSE.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

See Also: CloseUserDictionaries
CloseDictionaries

GoodWord Function

Applies to
ABaseSpl unit

Declaration
function GoodWord(DictPtr: pointer; TheWord : **string**) : boolean;

Description
The GoodWord function will return TRUE if the word in TheWord was found in either the main dictionary or any of the open user dictionaries. It will return FALSE if the word was not found in any of the open dictionaries or if there are currently no open dictionaries.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

If there are characters in the word that are not part of the allowed Character Set then they be stripped from word before it is tested.

Example:
```
if not ABaseSpl.GoodWord(MyComp.DictPtr, CurrentWord) then
    Form1.SuggestReplacements(CurrentWord);
```

GetUserDictionaryList Function

Applies to
ABaseSpl unit

Declaration
function GetUserDictionaryList(DictPtr: pointer; DictID : integer) : TStringList;

Description
The GetUserDictionary function will return a sorted TStringList containing all of the words in the specified user dictionary. The primary use for this method is in Deleting Words From The Dictionary.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

Example: DictForm.WordList.Items := ABaseSpl.GetUserDictionary(MyComp.DictPtr, UserDictID);

InitDictionaryData Procedure

Applies to
ABaseSpl unit

Declaration
procedure InitDictionaryData(var DictPtr : pointer);

Description
The InitDictionaryData procedure is one of the most important procedures in the base unit, the other is the ReleaseDictionaryData procedure. Before you use any of the other procedures or functions in the ABaseSpl unit your component must call the InitDictionaryData procedure to obtain a DictPtr pointer.

This is to allow the unit to deal with multiple instances of the component using it. The pointer returned will point to a structure used internally by the unit to store caching information, dictionary file information and so on. All the other procedures and functions require the pointer be passed to them in order to operate properly.

If the internal structure could not be created (due to no free memory) a NIL pointer will be returned.

The best place to call the InitDictionaryData procedure is within the Create method of the component you have built around the ABaseSpl unit.

Example:
```
ABaseSpl.InitDictionaryData(MyComp.DictPtr);
if MyComp.DictPtr <> nil then
  begin
    { continue initialization of component here }
  end;
```

OpenDictionary Function

Applies to
ABaseSpl unit

Declaration
function OpenDictionary(DictPtr: pointer; Filename : **string**) : boolean;

Description
The OpenDictionary function is used to open the main dictionary. You must include the full path as part of the filename. Only one main dictionary may be opened at a time. If the dictionary was opened successfully OpenDictionary will return TRUE. If the dictionary file could not be opened, is not a primary dictionary file (i.e. a user dictionary) or there is already a main dictionary open FALSE will be returned.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

See Also: OpenUserDictionary

Example:
```
if not ABaseSpl.OpenDictionary(MyComp.DictPtr, Form1.MainDictName) then
  Form1.OpeningDictionaryError;
```

OpenUserDictionary Function

Applies to
ABaseSpl unit

Declaration
function OpenUserDictionary(DictPtr: pointer; Filename : **string**) : integer;

Description
The OpenUserDictionary function is used to open an existing user dictionary. You must include the full path as part of the filename passed to it. If the dictionary was opened successfully, the value returned is the DictID for that dictionary and will be a positive number. It is important to remember the DictID returned as it is required to use the opened user dictionary with the AddWord, CloseUserDictionary, DeleteUserDictionary, and GetUserDictionary procedures and functions..

If the dictionary could not be opened (i.e. the file was not found or it was not a user dictionary file) -1 will be returned.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

See Also: AddWord
CloseUserDictionary
DeleteUserDictionary
GetUserDictionary

Example:
```
NewID := ABaseSpl.OpenUserDictionary(MyComp.DictPtr, Form1.UserDictName);
if NewID <> -1 then
  Form1.UserDictID := NewID
else
  Form1.OpenUserDictionaryError;
```

ReleaseDictionaryData Procedure

Applies to
ABaseSpl unit

Declaration
procedure ReleaseDictionaryData(var DictPtr : pointer);

Description
The ReleaseDictionaryData procedure is the other of the two most important procedures with the other being the InitDictionaryData procedure. The ReleaseDictionaryData procedure will release the memory assigned to the internal data structure used by the ABaseSpl unit. If any dictionaries are open they will also be closed.

If you do not call the ReleaseDictionaryData procedure before you component built around the ABaseSpl unit is destroyed the memory allocated will not be released.

The best place to place the call to ReleaseDictionaryData is in the Free method of the component you have built around ABaseSpl.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

SuggestCloseMatch Function

Applies to
ABaseSpl unit

Declaration
function SuggestCloseMatch(DictPtr: pointer; TheWord : **string**;
MaxSuggestions : byte) : TStringList;

Description
The SuggestCloseMatch function will create a list of suggested correct spellings for the word passed in TheWord. The number of suggestions returned is controlled by the MaxSuggestinos parameter. The limit for MaxSuggestions is 255.

The list will be order based on how close the method believes the spelling is the correct spelling of the word you passed it. The first word in the list will be the most likely, the second the second most likely and so on.

The SuggestCloseMatch method returns suggestions based on a Close Match style of suggestions.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

See Also: SuggestPhoneme

Example:
```
Form1.SuggestDialog.SugList.Items :=
    ABaseSpl.SuggestCloseMatch(MyComp.DictPtr, CurrentWord, 15);
```

SuggestPhoneme Function

Applies to
ABaseSpl unit

Declaration
function SuggestPhoneme(DictPtr: pointer; TheWord : **string**;
MaxSuggestions : byte) : TStringList;

Description
The SuggestPhoneme function will create a list of suggested correct spellings for the word passed in TheWord. The number of suggestions returned is controlled by the MaxSuggestions parameter. The limit for MaxSuggestions is 255.

The list will be order based on how close the method believes the spelling is the correct spelling of the word you passed it. The first word in the list will be the most likely, the second the second most likely and so on.

The SuggestPhoneme method returns suggestions based on a Phoneme Match style of suggestions.

The DictPtr must contain a valid pointer returned by the InitDictionaryData procedure. If an invalid pointer is passed results are unpredictable and will most likely result in an eventual system crash.

See Also: SuggestCloseMatch

Example:
```
Form1.SuggestDialog.SugList.Items :=
    ABaseSpl.SuggestPhoneme(MyComp.DictPtr, CurrentWord);
```

DictPtr Type

Applies to
ABaseSpl unit

The DictPtr parameter used by this unit is a normal untyped pointer. There is nothing magic about the name DictPtr, but it is recommended that you use it as the name of the variable to help avoid confusion since the help files all refer to it as the DictPtr parameter.

You must call the InitDictionaryData procedure to have ABaseSpl create the data structure used internally by the unit before passing it to any of the other procedures or functions in the ABaseSpl unit. Failure to do so will result in unpredictable results and eventually a system crash. Additionally, you must call the ReleaseDictionaryData procedure to release the memory allocated to the DictPtr when you are finished using the unit. If you do not call the ReleaseDictionaryData procedure the memory allocated by ABaseSpl for the DictPtr will not be freed.

The best place to call the InitDictionaryData procedure is within the Create method of the component you are designing to use to the ABaseSpl unit. The best place to call the ReleaseDictionaryData procedure is within the Free method of your component.

Character Set

Applies to

ABaseSpl unit
CheckOrph, MemoSpell, AcropSpell components

Description

The allowed characters in the dictionary are:

Aa Bb Cc Dd Ee Ff Gg Hh Ii Jj Kk Ll Mm
Nn Oo Pp Qq Rr Ss Tt Uu Vv Ww Xx Yy Zz
Àà Áá Ââ Ãã Ää Åå Ææ Çç Œœ Èè Éé Êê Ëë
Ìì Íí Îî Ïï Ññ Òò Óó Ôô Õõ Öö Šš Ùù Úú
Ûû Üü Ýý Ÿÿ ß - ' `

Deleting Words From The Dictionary

Applies to

BaseASpl unit
AcropSpell component
BsASpl32 unit
Acrops32 component

Due to the graph structure used to store the dictionary files you cannot simply delete a simple word from a user dictionary file. However, there is a way to remove words from the dictionary and as a size effect have the size of the dictionary file reduced.

The method of removing words from the dictionary consists of obtaining the complete list of words in a user dictionary, adding an removing the words from the list and then building a new dictionary. This is not as major a drawback as if might first seem since normally you would be presenting the user of your application with a complete list of the words to manipulate before deleting the words.

The steps involve the following:

1. Using the GetUserDictionaryList function in the ABaseSpl unit or GetUserDictionary function in the AcropSpell component.
2. Allow the user to add words to the list or delete words from the list.
3. Delete the dictionary using the DeleteUserDictionary function from the ABaseSpl unit or AcropSpell component.
4 Use the BuildUserDictionary function from either the ABaseSpl unit or AcropSpell component to create a new dictionary.

The most complicated part is involved in step 2. However, Delphi makes creating a dialog form to manipulate the list fairly easy.

There is one advantage to this method of deleting words from the dictionary: Since the TStringList you provide the BuildUserDictionary file gets sorted first the spell checking engine within ABaseSpl can use a more efficient method for adding words to the dictionary file resulting in a (normally) smaller file.