Spelmate Help

Spelmate for Windows

Version 1.1

This Help File was produced using Help Edit from Aciran Software

Contents

Glossary

What is Spelmate?

Spelmate is a Windows DLL (Dynamic Link Library) that allows you to add Spell checking to your Windows Application.

Spelmate is available in both British and American versions.

How do I use it?

There are two basic ways of using any DLL, you can create and link in an Import library, or you can explicitly load the DLL.

Import Library

The simplest method is to create an import library or in the case of Turbo Pascal, an import Unit.

If you are using Borland C++ you can create an import library automatically using IMPLIB on the DLL.

To create a Turbo Pascal Import Unit use the source file Spell.Pas, part of which is shown below. You will have to change the Units in the Uses section if you are using TPW, as the units listed are for Borland Pascal 7.0.

```
function SpelMateInit : integer;
function Spellcheck(Word : PChar) : Boolean;
procedure AddWord(Word : PChar);
procedure IgnoreWord(Word : PChar);
function SuggestWord(Word : PChar) : Pchar;

implementation

function SpelmateInit;external 'SpelMate' index 1;
function SpellCheck; external 'Spelmate' index 2;
procedure AddWord;external 'spelmate' index 3;
procedure IgnoreWord;external 'spelmate' index 4;
function SuggestWord;external 'spelmate' index 5;
end.
```

This import unit identifies the main functions and procedures in Spelmate.dll.

Click on each function name for a description of it's use.

Load DLL explicitly

This method of using a DLL is more complicated, but does give you more control over how the DLL is used. The DLL will only be loaded when requested, and this can result in more efficient use of Windows resources.

In order to do this in Borland Pascal you must create procedural data types that describe the subroutine formats. These are shown below.

```
type
  TSpelmateInit = function :integer;
  TSpellCheck = function (AWord:Pchar):Boolean;
  TSuggestWord = function(AWord:PChar):PChar;
```

The declarations do not have identifies, only the key word procedure or function followed by any parameters and, for functions, a return type. Borland Pascal and TPW associates these declarations with the data defined, and you can then use variables of those types to call the DLL's routines. To do that you need to call LoadLibrary as shown:

function LoadLibrary(LibFileName : PChar) : THandle;

After calling LoadLibrary, but before calling FreeLibrary, you can search for the subroutines in the DLL by calling GetProcAddress. In order to save the result, declare a variable of the procedure or function datatype created earlier.

```
{make instances of the functions from our prototypes}
var
SpelmateInit : TSpelmateInit;
SpellCheck : TSpellCheck;
SuggestWord : TSuggestWord;

  LoadCursor(0,IDC_WAIT);
  Handle := LoadLibrary('Spelmate.Dll');
  LoadCursor(0,IDC_ARROW);
  if Handle < 32 then    {if failed to load/find DLL inform user}
  begin
   MessageBeep(0);
   MessageBox(HWindow,'Unable to Load Spelling Dll','Application Error',MB_OK or
   MB_ICONSTOP);
   Exit;
  end;
  {Set our function addresses up to those in the DLL, should check really
   that they are not nil, as this indicates an error}
  @SpelmateInit := GetProcAddress(Handle,'SpelmateInit');
  @SpellCheck := GetProcAddress(Handle,'Spellcheck');
  @SuggestWord := GetProcAddress(Handle,'SuggestWord');
  {This dialog is just to keep the user happy while the main dictionary is
   being loaded as this can take a few seconds}
  Dialog :=New(PDlgWindow,Init(nil,'LoadSpell'));
  Application^.MakeWindow(Dialog);
  {Ask the DLL to init, and get back the status information}
  result := SpelmateInit;
```

```
{remove the dialog now DLL loaded}
Dialog^.done;
{if result = -1 then OK}
if result <> -1 then
begin
   MessageBox(HWindow,'Cannot Access Spelling Dictionary','Application Error',MB_OK or MB_ICONSTOP);
   Exit;
{ if result not -1 then an error occurred, and so do not spell check}
{ Possible error codes are }
{ 0 not enough memory }
{ 1 Main dictionary not found}
{ 2 Stream access error, main dictionary}
{ 3 Initialisation error, main dictionary}
{ 4 Read error, main dictionary}
{ 5 Corrupt file or wrong file type, error, main dictionary}
{ 6 Stream access error, private dictionary}
{ 7 Initialisation error, private dictionary}
{ 8 Read error, private dictionary}
{ 9 Corrupt file or wrong file type, error, private dictionary}
end;
```

The above shows a section of code from one of the example programs on disk. It illustrates not only how to load the DLL and get access to it's subroutines, but also how to handle errors and how to keep the user informed, especially as it can take several seconds to load the main dictionary, and the user may think something has failed if there was no message to say what was happening.

Once we are finished using the Dll, then we should free it when done, as shown below:

```
FreeLibrary(Handle); {release Spelmate DLL, or you may wait until your program exits, and do it in your destructor}
```

The procedures and functions created above can be called just like any standard functions, provided the DLL loaded successfully.

What Functions are available?

Spelmate provides the following functions and procedures:

function SpelmateInit; Used to initialise the DLL and load the main Dictionary and private dictionary if it exists. You **must** call this function before any other.

function SpellCheck; Used to check the spelling of a word.

procedure AddWord; Used to add a word to the users private dictionary.

procedure IgnoreWord; Adds the word to a temporary dictionary so that all future occurrences will be ignored.

function SuggestWord; Used to suggest the correct spelling of an unknown word.

Click on each procedure or function name for more information.

Normally your Application will call SpelmateInit once to ensure that it is initialised, and then alternate between calling Spelcheck and SuggestWord as needed.

function SpelmateInit.

This function must be called to setup and initialise the DLL. The format is:

result := SpelmateInit; { Pascal call }

Where result is a variable of type integer; The return value should be checked before performing any DLL operations. The range of values is shown below:

```
{ -1 No Error   }
{ 0 not enough memory }
{ 1 Main dictionary not found}
{ 2 Stream access error, main dictionary}
{ 3 Initialisation error, main dictionary}
{ 4 Read error, main dictionary}
{ 5 Corrupt file or wrong file type, error, main dictionary}
{ 6 Stream access error, private dictionary}
{ 7 Initialisation error, private dictionary}
{ 8 Read error, private dictionary}
{ 9 Corrupt file or wrong file type, error, private dictionary}
```

The first time the DLL is Loaded or called, a flag is set to say that it has not yet being setup. When SpelmateInit is called, the main and private dictionaries are loaded, and this flag is cleared. Subsequent calls to SpelmateInit will result in the return value -1, provided all went well during initialisation. It will not waste time re-loading the dictionaries. It the start up failed on a previous attempt, Spelmate will re-try, assuming that the user has made an attempt to correct any faults, e.g. missing files.

This approach has been taken to maximise the success of the DLL loading, or at least you application getting up and running, and being able to inform the user if something is wrong, like they deleted the main dictionary by mistake last session, or forgot to put the spell checker on their path.

function SpellCheck.

Use this function to check the spelling of words. the format is:

result := spellcheck(AWord) ;

where result is a variable of type boolean, and AWord is a PChar variable;

The value of result is true if the word was found, either in the main, private or ignoreAll dictionaries, or else it is false meaning the word is unknown.

procedure AddWord.

Use this procedure to add the word to the users private dictionary. The maximum number of words allowed is 16,384.

The format is:

AddWord(AWord);

where AWord is a PChar variable;

Procedure IgnoreWord.

If the word is to be ignored, and is likely to be repeated, use this procedure to place it into a temporary dictionary, so that all future occurrences will be ignored.

The format is:

IgnoreWord(AWord);

where AWord is a PChar variable;

function SuggestWord.

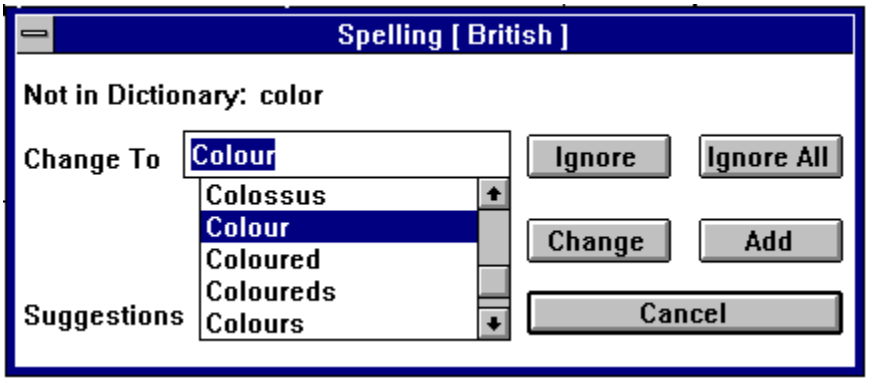
Use this function to suggest the possible correct spelling of the word.

The format is:

NewWord := SuggestWord(AWord);

where NewWord and AWord are both PChar variables.

When this function is called Spelmate will present the user with the dialog box shown below.



Click on each of the user controls for more information.

What Languages does it support?

Spelmate can be called from any programming language that supports Windows DLL's, this includes C++, Pascal (Borland and TPW), Visual Basic and others such as ObjectVision.

The Sample Programs

Sample programs are provided in three languages.

C Sample Programs
Pascal Sample Programs
Visual Basic Sample programs

Select either of the above for more information

C Sample Programs

Special thanks to Stewart McSporran of MEDC for this C example.

The C example is called SPELLTST.EXE and was written using Borland C++.

This is a simple text editor with spell checking facilities and is based on the Borland FileApp example. Spelltst uses an Import library SPELMATE.LIB to access Spelmate DLL. The import library was created using IMPLIB. It also uses the module Spell.cpp which has string and spell checking functions to spell check the complete text in an edit control.

All source files to make SPELLTST.EXE are included, and Stewart has liberally commented the source code.

SPELLTST makes use of a new feature added to Spelmate V1.1. The ability to determine which half of the screen displays the SuggestWord dialog box.

Murphy's law has it that when ever the spell checking dialog box pops up on the screen, it always manages to obscure the part of the text containing the word being checked. As a default, Spelmate now places the dialog box in the bottom centre of the screen. However, if the procedure **DisplayAtTop** is called first, the dialog box will appear at the top centre of the screen.

All your Application has to do is work out where on the screen the word is, and then put the dialog box in the other half. How this is done is shown in SPELLTST (and in SPELCHEK.EXE in the pascal example).

Pascal Sample Programs

Two sample Pascal programs are included to illustrate the use of Spelmate. Both were written using Borland Pascal 7.0, and full source code is included.

The first program, SPELCHEK.EXE is a simple file spell checking program. It parses the text file selected by the user, and extracts words for spell checking. Words are defined as a series of one or more characters in the range [a..z,A..Z].

When it has a word it calls the function Spelcheck to see it it is a valid word. If the word is not found then it calls SuggestWord. The users choice of suggestion is then displayed in a message box, (this may be the original word if no change was made).

The program does not attempt anything else. It is used for illustration purposes only. It uses the import method of loading the DLL.

The second program, SpellEdit is a lot more sophisticated. It uses the LoadLibrary function to access the DLL.

SpellEdit is based on the Borland demo program FileApp, which is a simple text Editor program. However some changes were made to the basic Edit control, so that a handle could be obtained to the Edit control's text in order to scan it.

The Editor had a menu option for spell checking, and when selected it will start parsing the text from the current cursor position until the end of the text, or until the user selects cancel. When if encounters an unknown word, it will highlight the word in the Edit control, and call SuggestWord. If the users changes the word, the new word will replace the old one. The user can save the updated text when spell checking is complete.

Murphy's law has it that when ever the spell checking dialog box pops up on the screen, it always manages to obscure the part of the text containing the word being checked. As a default, Spelmate now places the dialog box in the bottom centre of the screen. However, if the procedure **DisplayAtTop** is called first, the dialog box will appear at the top centre of the screen.

All your Application has to do is work out where on the screen the word is, and then put the dialog box in the other half. How this is done is shown in SPELCHEK (and in SPELLTST.EXE in the C example).

Visual Basic Examples

Special thanks to Alistair McMonnies of MEDC for these VB examples

Alistair has supplied two Visual Basic examples. These are VBSPLMIN.EXE and SPELTEST.EXE.

VBSPLMIN.EXE as the name suggests is a minimum VB example showing how to access Spelmate from VB. It consists of a simple Edit control in a Form, and all the user has to do is enter some word and then press return in order to have the spelling of the word checked.

SPELTEST.EXE is along the same lines as the C and Pascal programs in that it is a complete Editor with spell checking features. Spelmate can be linked in to Visual Basic programs extremely easily, as Alistair will tell you in his own words.

Read the VB Notes by clicking on the VB Notes icon in the Spelmate Group.

Glossary

DLL (Dynamic link Library)
Help Edit
IMPLIB (Import Librarian)
RTF (Rich Text Format)
Shareware

Dynamic Link Library.

A Dynamic Link Library (DLL) is a Windows library of functions and procedures that are linked to program at run time, as opposed to linking to a program at compile time. The result is that many programs can share common code, thus making more efficient use of Windows resources.

Help Edit

Help Edit is a simple to use Windows Tool that is used to create Windows Help Files, in conjunction with the Microsoft Help Compiler. It removes the need for an RTF (Rich Text Format) word processor, such as Word for Windows. Help Edit handles all jump and definition labels, and supports multiple fonts and colours, as well as Bitmaps. Help Edit was the first program to make use of Spelmate. This Help File was written using Help Edit.

RTF

RTF (Rich Text Format) is a word processor format developed by Microsoft in 1987. It allows the complete description of a page layout in a document, including fonts, colour and graphics, to be express in a purely textual form, similar in some ways to a PostScript file. It is only supported by high end word processors such as Microsoft Word.

IMPLIB

IMPLIB is a Borland program that is used to create an import library automatically from an existing DLL. Simple run IMPLIB and select the DLL, and IMPLIB will create an import library with the same name as the DLL, but with the extension .LIB. You can then add this library name to your project, and the linker will generate code to automatically load the DLL when your program starts. This is the same as the import Unit Spell created for our example.

Shareware

Shareware is a form of try before you buy. It is NOT a means of obtaining free software. Shareware is not free. If you find a Shareware program useful and intent to continue using it you are obliged to register it with the author, and pay for it.

If the word is correct, but unknown to Help Edit, you can add the word to your private dictionary.

If the word is correct, but not important enough to keep, eg. a post or zip code, you can simply ignore it.

If it is likely that the word will be repeated, you can select Ignore All, and Spelmate will place the word in a Temporary dictionary, and will ignore all future Occurrences.

If the Edit control box of the suggestion list contains the word you want, you can select change and Help Edit will replace the old word in your text with this new word.

Select Cancel to end the spell check, the cursor will be placed at the point spell checking started.

This is the word the spell checker did not recognise in your text.

The spell checker will offer a list of words from the dictionary that is closest to the word it does not recognise.