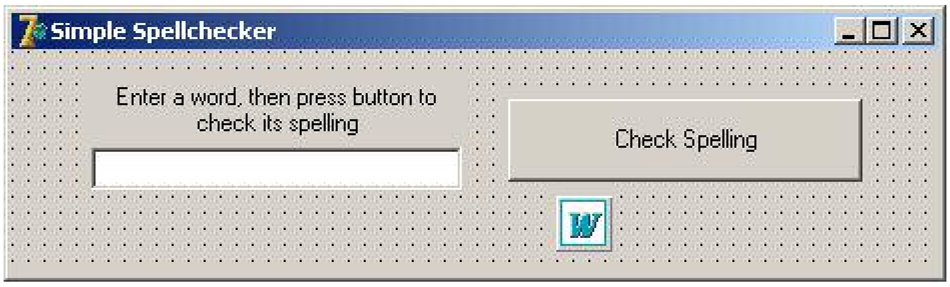
Spellchecking from within Delphi

This problem arose from a project in which groups of characters were produced; the question was to determine whether any particular group was actually a valid word. Since any standard word processor has a spellcheck facility, it would seem sensible to access that facility within the program, rather than trying to produce a specialised dictionary just for the purposes of the program.

In fact, using MS Word's spellchecking procedure is so simple that it would be ridiculous to attempt anything else if you have MS Word installed on your computer.

Example 1 - a simple spellchecker

Create a new application. On the form, put a TEdit (called edinput), a TButton (called edCheckSpelling) and a TWordApplication (from the servers tab). A TLabel suitably captioned may be appropriate too:

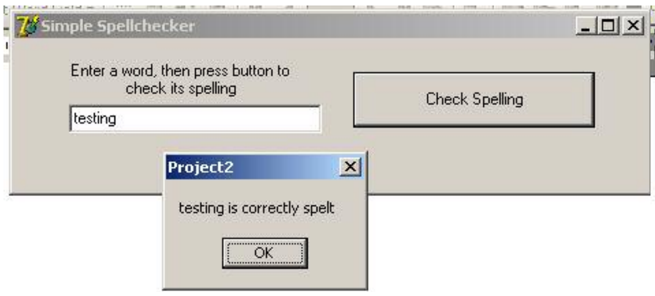


This will allow the user to enter a word into the edit box, then press the button to check its spelling. Delphi will display an appropriate message.

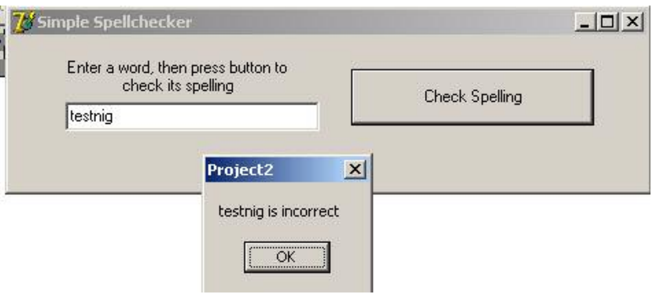
The only coding that should need doing is the button's OnClick method:

```
procedure TForm1.btnCheckSpellingClick(Sender: TObject);
begin
   if WordApplication1.CheckSpelling(edinput.Text)
    then showmessage(edinput.Text + ' is correctly spelt')
     else showmessage(edinput.Text+' is incorrect');
end;
```

Only one instruction needs writing. Having put the WordApplication on the form, it doesn't need any other attention; merely placing it there makes Word's CheckSpelling method available. CheckSpelling has several different formats, the simplest of which is to take a single parameter which is the string to be checked. CheckSpelling is a Boolean function, which returns True if the word is found in the dictionary, and False otherwise.



testing is correctly spelt



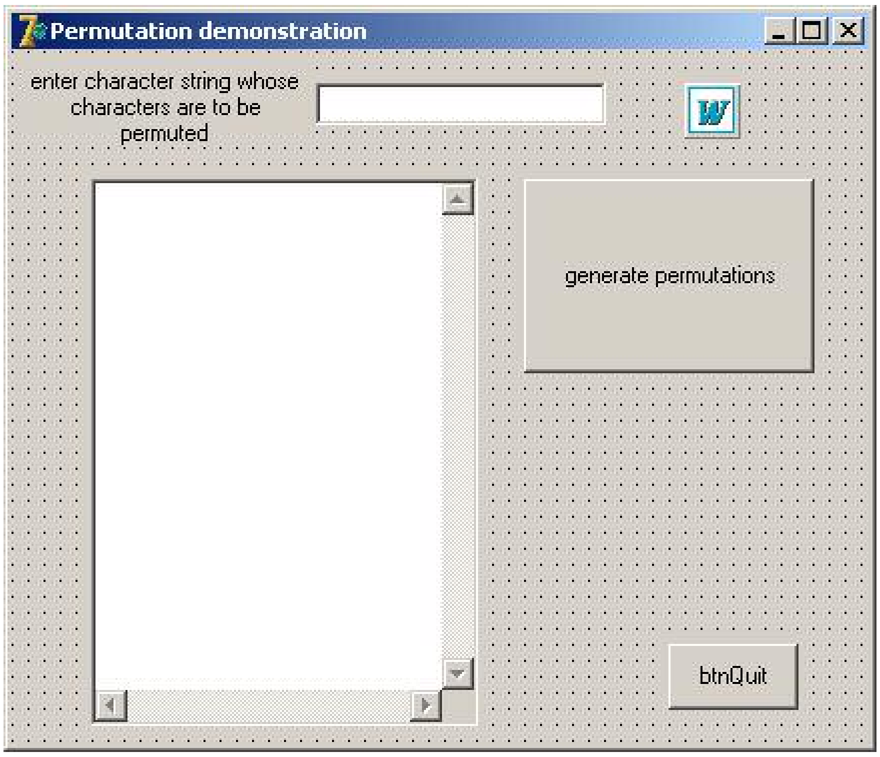
testnig is incorrect

Note that there is much more available, such as a list of suggested alternatives. To find out more, see the separate article on outputting from Delphi to MS Word, or:
http://delphi.about.com/od/kbcontrolole/l/aa032701a.htm
or do a Google search on "Spellcheck Word Delphi".

Example 2 - Anagrams

This example allows the user to enter a series of letters. The program then creates all possible permutations of those letters, and spellchecks each one. Those that are genuine words are listed to a memo box.

Create a new application and save it to a new folder.


Permutation demonstration
enter character string whose characters are to be permuted
generate permutations
btnQuit

Add to the form two buttons (btnQuit, btnPerm), an edit box (edinput), a memo box, and a TWordApplication, plus a label as shown.

The idea is that you can enter a word or series of letters into the edit box. On pressing the button btnPerm, the program will generate all permutations of the entered letters.

Permutations - an example of Recursive programming

Consider the string of characters 1234. If you were asked to write down the permutations of these characters (or numbers), you would probably write down (if you thought about it logically):

```
1234
1243
1324
1342
1423
1432
then    2134
        2143
        2314
        2341
        2413
        2431
        then    3124    etc.
```

In other words, you are saying that to find all the permutations of 4 characters, you take each character in turn, and for each of them, you find all the permutations of the remaining 3 characters. Similarly, to find all the permutations of those 3 characters, you take each of the three in turn, and find the permutations of the remaining 2 characters, etc.

This is standard material for a recursive definition.

```
Permute(list of n items) =
     for counter:= 1 to n do
       item[counter} + Permute(list of n-1 items, where item[counter] is removed from list of n items)
```

As with all recursive routines, there needs to be an exit option. This occurs when n has the value 1, i.e. there is only one permutation, and this can then lead to an output.

The number of loops in the top level will be n; that in the next level will be n-1 etc, which means that the number of permutations is n(n-1)(n-2)….1, which is factorial n.

In this example, the list of n items is a list of characters, i.e. a string. This is the entire unit code:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls, OleServer, WordXP;
```

Added by Delphi when the TWordApplication is put on the form

```
type
  TForm1 = class(TForm)
    Memo1: TMemo;
    edInput: TEdit;
    btnPerm: TButton;
    Label1: TLabel;
    btnQuit: TButton;
    WordApplication1: TWordApplication;
    procedure btnPermClick(Sender: TObject);
    procedure btnQuitClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}
procedure outputpermutation(s: string);
begin
     if form1.WordApplication1.CheckSpelling(s)
       then form1.Memo1.Lines.Add(s);
end;
```

`outputpermutation(s)` is declared as a separate procedure outside Form1's class definition. Therefore, WordApplication1 and Memo1, will not be recognised within outputpermutation unless their full names, including form1, are given.

```
procedure permute(s_in :string; n:integer; s_on:string);
var i:integer; s_temp:string;
begin
  if n>1 then
    begin
      for i:=1 to n do
       begin
         s_on:=s_on + s_in[i];
         if i=1 then s_temp:=copy(s_in,2,length(s_in)-1)
          else s_temp:=copy(s_in,1,i-1) + copy(s_in,i+1,length(s_in)-i);
         permute(s_temp,n-1,s_on);
         s_on:=copy(s_on,1,length(s_on)-1);
       end;
    end else
    begin
      outputpermutation(s_on+s_in);
    end;
end;
```

s_in is the string to be permuted. n is the number of characters in it, and s_on is the substring to be passed on to the next level of recursion, i.e. it contains the left side of the string before those being presented for further recursion.

By using a separate procedure to output the permuted string, you can control *how* it is output more conveniently - see below.

```
procedure TForm1.btnPermClick(Sender: TObject);
var s_in : string;
begin
 memo1.Clear;
 s_in:=edInput.Text;
 permute(s_in,length(s_in),'');
end;

procedure TForm1.btnQuitClick(Sender: TObject);
begin
  application.Terminate;
end;

end.
```

This initiates the permutation process by clearing the memo box, grabbing the contents of the edit box, and calling permute at the first level.

In the above example, the procedure `outputpermutation` causes only valid words to be output to the memo box. If instead the code was simplified to:

```
procedure outputpermutation(s: string);
begin
     form1.Memo1.Lines.Add(s);
end;
```

this would cause all the permutations to appear in the memo box. You may like to try this first to see how it works. It would also be instructive to produce a trace table for the program if you are unsure of its operation. You could alternatively use Delphi's debugging facilities - set up watches (Run menu, Add Watch) for s_on, s_in, n, i, and observe how these change as you single-step (by pressing F7) through the program. You may find it convenient to create a breakpoint next to the 2nd or 3rd instruction of btnPerm's code, then allow the program to run to this point before using F7.