Simulate the Visual Basic SendKeys and AppActivate routines in Delphi

This unit includes two routines that simulate popular Visual Basic routines: *Sendkeys* and *AppActivate*. *SendKeys* takes *PChar* as its first parameter and a *Boolean* as its second, like so:

```
SendKeys('KeyString', Wait);
```

where *KeyString* is a string of key names and modifiers that you want to send to the current input focus and *Wait* is a Boolean variable or value that indicates whether *SendKeys* should wait for each key message to be processed before proceeding. See the table below for more information.

AppActivate also takes a *PChar* as its only parameter, like so:

```
AppActivate('WindowName');
```

where *WindowName* is the name of the window that you want to make the current input focus.

SendKeys supports the Visual Basic *SendKeys* syntax, as documented below.

Supported modifiers	
+	Shift
^	Control
%	Alt

Surround sequences of characters or key names with parentheses in order to modify them as a group. For example, '+abc' shifts only 'a', while '+(abc)' shifts all three characters.

Supported special characters	
~	Enter
(	Begin modifier group (see above)
)	End modifier group (see above)
{	Begin key name text (see below)
}	End key name text (see below)

Any character that can be typed is supported. Surround the modifier keys listed above with braces in order to send as normal text.

Supported key names (surround these with braces):

```
BKSP, BS, BACKSPACE
BREAK
CAPSLOCK
CLEAR
DEL
DELETE
DOWN
END
ENTER
ESC
ESCAPE
F1
F2
F3
```

F4
F5
F6
F7
F8
F9
F10
F11
F12
F13
F14
F15
F16
HELP
HOME
INS
LEFT
NUMLOCK
PGDN
PGUP
PRTSC
RIGHT
SCROLLLOCK
TAB
UP

Follow the keyname with a space and a number to send the specified key a given number of times (e.g., {left 6}).

Here's the unit.

```
unit sndkey32;

interface

Uses SysUtils, Windows, Messages;

Function SendKeys(SendKeysString : PChar; Wait : Boolean) : Boolean;
function AppActivate(WindowName : PChar) : boolean;

{Buffer for working with PChar's}

const
  WorkBufLen = 40;
var
  WorkBuf : array[0..WorkBufLen] of Char;

implementation
type
  THKeys = array[0..pred(MaxLongInt)] of byte;
var
  AllocationSize : integer;

(*
Converts a string of characters and key names to keyboard events and
passes them to Windows.

Example syntax:

SendKeys('abc123{left}{left}{left}def{end}456{left 6}ghi{end}789', True);
*)

Function SendKeys(SendKeysString : PChar; Wait : Boolean) : Boolean;
type
  WBytes = array[0..pred(SizeOf(Word))] of Byte;
```

```
  TSendKey = record
    Name : ShortString;
    VKey : Byte;
  end;

const
  {Array of keys that SendKeys recognizes.
  If you add to this list, you must be sure to keep it sorted alphabetically
  by Name because a binary search routine is used to scan it.}

  MaxSendKeyRecs = 41;
  SendKeyRecs : array[1..MaxSendKeyRecs] of TSendKey =
  (
   (Name:'BACKSPACE';       VKey:VK_BACK),
   (Name:'BKSP';            VKey:VK_BACK),
   (Name:'BREAK';           VKey:VK_CANCEL),
   (Name:'BS';              VKey:VK_BACK),
   (Name:'CAPSLOCK';        VKey:VK_CAPITAL),
   (Name:'CLEAR';           VKey:VK_CLEAR),
   (Name:'DEL';             VKey:VK_DELETE),
   (Name:'DELETE';          VKey:VK_DELETE),
   (Name:'DOWN';            VKey:VK_DOWN),
   (Name:'END';             VKey:VK_END),
   (Name:'ENTER';           VKey:VK_RETURN),
   (Name:'ESC';             VKey:VK_ESCAPE),
   (Name:'ESCAPE';          VKey:VK_ESCAPE),
   (Name:'F1';              VKey:VK_F1),
   (Name:'F10';             VKey:VK_F10),
   (Name:'F11';             VKey:VK_F11),
   (Name:'F12';             VKey:VK_F12),
   (Name:'F13';             VKey:VK_F13),
   (Name:'F14';             VKey:VK_F14),
   (Name:'F15';             VKey:VK_F15),
   (Name:'F16';             VKey:VK_F16),
   (Name:'F2';              VKey:VK_F2),
   (Name:'F3';              VKey:VK_F3),
   (Name:'F4';              VKey:VK_F4),
   (Name:'F5';              VKey:VK_F5),
   (Name:'F6';              VKey:VK_F6),
   (Name:'F7';              VKey:VK_F7),
   (Name:'F8';              VKey:VK_F8),
   (Name:'F9';              VKey:VK_F9),
   (Name:'HELP';            VKey:VK_HELP),
   (Name:'HOME';            VKey:VK_HOME),
   (Name:'INS';             VKey:VK_INSERT),
   (Name:'LEFT';            VKey:VK_LEFT),
   (Name:'NUMLOCK';         VKey:VK_NUMLOCK),
   (Name:'PGDN';            VKey:VK_NEXT),
   (Name:'PGUP';            VKey:VK_PRIOR),
   (Name:'PRTSC';           VKey:VK_PRINT),
   (Name:'RIGHT';           VKey:VK_RIGHT),
   (Name:'SCROLLLOCK';      VKey:VK_SCROLL),
   (Name:'TAB';             VKey:VK_TAB),
   (Name:'UP';              VKey:VK_UP)
  );

  {Extra VK constants missing from Delphi's Windows API interface}
  VK_NULL=0;
  VK_SemiColon=186;
  VK_Equal=187;
  VK_Comma=188;
  VK_Minus=189;
  VK_Period=190;
  VK_Slash=191;
  VK_BackQuote=192;
  VK_LeftBracket=219;
  VK_BackSlash=220;
  VK_RightBracket=221;
  VK_Quote=222;
  VK_Last=VK_Quote;

  ExtendedVKeys : set of byte =
  [VK_Up,
```

```
   VK_Down,
   VK_Left,
   VK_Right,
   VK_Home,
   VK_End,
   VK_Prior,  {PgUp}
   VK_Next,   {PgDn}
   VK_Insert,
   VK_Delete];

const
  INVALIDKEY = $FFFF {Unsigned -1};
  VKKEYSCANSHIFTON = $01;
  VKKEYSCANCTRLON = $02;
  VKKEYSCANALTON = $04;
  UNITNAME = 'SendKeys';
var
  UsingParens, ShiftDown, ControlDown, AltDown, FoundClose : Boolean;
  PosSpace : Byte;
  I, L : Integer;
  NumTimes, MKey : Word;
  KeyString : String[20];

procedure DisplayMessage(Message : PChar);
begin
  MessageBox(0,Message,UNITNAME,0);
end;

function BitSet(BitTable, BitMask : Byte) : Boolean;
begin
  Result:=ByteBool(BitTable and BitMask);
end;

procedure SetBit(var BitTable : Byte; BitMask : Byte);
begin
  BitTable:=BitTable or Bitmask;
end;

Procedure KeyboardEvent(VKey, ScanCode : Byte; Flags : Longint);
var
  KeyboardMsg : TMsg;
begin
  keybd_event(VKey, ScanCode, Flags,0);
  If (Wait) then
    While (PeekMessage(KeyboardMsg,0,WM_KEYFIRST, WM_KEYLAST, PM_REMOVE)) do
    begin
      TranslateMessage(KeyboardMsg);
      DispatchMessage(KeyboardMsg);
    end;
end;

Procedure SendKeyDown(VKey: Byte; NumTimes : Word; GenUpMsg : Boolean);
var
  Cnt : Word;
  ScanCode : Byte;
  NumState : Boolean;
  KeyBoardState : TKeyboardState;
begin
  if (VKey=VK_NUMLOCK) then
  begin
    NumState:=ByteBool(GetKeyState(VK_NUMLOCK) and 1);
    GetKeyBoardState(KeyBoardState);
    If NumState then
      KeyBoardState[VK_NUMLOCK]:=(KeyBoardState[VK_NUMLOCK] and not 1)
    else
      KeyBoardState[VK_NUMLOCK]:=(KeyBoardState[VK_NUMLOCK] or 1);
    SetKeyBoardState(KeyBoardState);
    Exit;
  end;

  ScanCode:=Lo(MapVirtualKey(VKey,0));
  For Cnt:=1 to NumTimes do
    If (VKey in ExtendedVKeys) then
    begin
```

```
      KeyboardEvent(VKey, ScanCode, KEYEVENTF_EXTENDEDKEY);
      If (GenUpMsg) then
        KeyboardEvent(VKey, ScanCode, KEYEVENTF_EXTENDEDKEY or KEYEVENTF_KEYUP)
    end
    else
    begin
      KeyboardEvent(VKey, ScanCode, 0);
      If (GenUpMsg) then KeyboardEvent(VKey, ScanCode, KEYEVENTF_KEYUP);
    end;
end;

Procedure SendKeyUp(VKey: Byte);
var
  ScanCode : Byte;
begin
  ScanCode:=Lo(MapVirtualKey(VKey,0));
  If (VKey in ExtendedVKeys)then
    KeyboardEvent(VKey, ScanCode, KEYEVENTF_EXTENDEDKEY and KEYEVENTF_KEYUP)
  else
    KeyboardEvent(VKey, ScanCode, KEYEVENTF_KEYUP);
end;

Procedure SendKey(MKey: Word; NumTimes : Word; GenDownMsg : Boolean);
begin
  If (BitSet(Hi(MKey),VKKEYSCANSHIFTON)) then
    SendKeyDown(VK_SHIFT,1,False);
  If (BitSet(Hi(MKey),VKKEYSCANCTRLON)) then
    SendKeyDown(VK_CONTROL,1,False);
  If (BitSet(Hi(MKey),VKKEYSCANALTON)) then
    SendKeyDown(VK_MENU,1,False);
  SendKeyDown(Lo(MKey), NumTimes, GenDownMsg);
  If (BitSet(Hi(MKey),VKKEYSCANSHIFTON)) then
    SendKeyUp(VK_SHIFT);
  If (BitSet(Hi(MKey),VKKEYSCANCTRLON)) then
    SendKeyUp(VK_CONTROL);
  If (BitSet(Hi(MKey),VKKEYSCANALTON)) then
    SendKeyUp(VK_MENU);
end;

{Implements a simple binary search to locate special key name strings}

Function StringToVKey(KeyString : ShortString) : Word;
var
  Found, Collided : Boolean;
  Bottom, Top, Middle : Byte;
begin
  Result:=INVALIDKEY;
  Bottom:=1;
  Top:=MaxSendKeyRecs;
  Found:=false;
  Middle:=(Bottom+Top) div 2;
  Repeat
    Collided:=((Bottom=Middle) or (Top=Middle));
    If (KeyString=SendKeyRecs[Middle].Name) then
    begin
       Found:=True;
       Result:=SendKeyRecs[Middle].VKey;
    end
    else
    begin
       If (KeyString>SendKeyRecs[Middle].Name) then
        Bottom:=Middle
       else
        Top:=Middle;
       Middle:=(Succ(Bottom+Top)) div 2;
    end;
  Until (Found or Collided);
  If (Result=INVALIDKEY) then
    DisplayMessage('Invalid Key Name');
end;

procedure PopUpShiftKeys;
begin
  If (not UsingParens) then
```

```
  begin
    If ShiftDown then SendKeyUp(VK_SHIFT);
    If ControlDown then SendKeyUp(VK_CONTROL);
    If AltDown then SendKeyUp(VK_MENU);
    ShiftDown:=false;
    ControlDown:=false;
    AltDown:=false;
  end;
end;

begin
  AllocationSize:=MaxInt;
  Result:=false;
  UsingParens:=false;
  ShiftDown:=false;
  ControlDown:=false;
  AltDown:=false;
  I:=0;
  L:=StrLen(SendKeysString);
  If (L>AllocationSize) then L:=AllocationSize;
  If (L=0) then Exit;

  While (I<L) do
  begin
    case SendKeysString[I] of
    '(' : begin
            UsingParens:=True;
            Inc(I);
          end;
    ')' : begin
            UsingParens:=False;
            PopUpShiftKeys;
            Inc(I);
          end;
    '%' : begin
             AltDown:=True;
             SendKeyDown(VK_MENU,1,False);
             Inc(I);
          end;
    '+' :  begin
             ShiftDown:=True;
             SendKeyDown(VK_SHIFT,1,False);
             Inc(I);
           end;
    '^' :  begin
             ControlDown:=True;
             SendKeyDown(VK_CONTROL,1,False);
             Inc(I);
           end;
    '{' : begin
            NumTimes:=1;
            If (SendKeysString[Succ(I)]='{') then
            begin
              MKey:=VK_LEFTBRACKET;
              SetBit(Wbytes(MKey)[1],VKKEYSCANSHIFTON);
              SendKey(MKey,1,True);
              PopUpShiftKeys;
              Inc(I,3);
              Continue;
            end;
            KeyString:='';
            FoundClose:=False;
            While (I<=L) do
            begin
              Inc(I);
              If (SendKeysString[I]='}') then
              begin
                FoundClose:=True;
                Inc(I);
                Break;
              end;
              KeyString:=KeyString+Upcase(SendKeysString[I]);
            end;
            If (Not FoundClose) then
```

```
            begin
              DisplayMessage('No Close');
              Exit;
            end;
            If (SendKeysString[I]='}') then
            begin
              MKey:=VK_RIGHTBRACKET;
              SetBit(Wbytes(MKey)[1],VKKEYSCANSHIFTON);
              SendKey(MKey,1,True);
              PopUpShiftKeys;
              Inc(I);
              Continue;
            end;
            PosSpace:=Pos(' ',KeyString);
            If (PosSpace<>0) then
            begin
              NumTimes := StrToInt(
                Copy(KeyString,Succ(PosSpace),Length(KeyString)-PosSpace)
              );
              KeyString:=Copy(KeyString,1,Pred(PosSpace));
            end;
            If (Length(KeyString)=1) then
              MKey:=vkKeyScan(KeyString[1])
            else
              MKey:=StringToVKey(KeyString);
            If (MKey<>INVALIDKEY) then
            begin
              SendKey(MKey,NumTimes,True);
              PopUpShiftKeys;
              Continue;
            end;
          end;
    '~' : begin
            SendKeyDown(VK_RETURN,1,True);
            PopUpShiftKeys;
            Inc(I);
          end;
    else
      begin
        MKey:=vkKeyScan(SendKeysString[I]);
        if (MKey<>INVALIDKEY) then
        begin
          SendKey(MKey,1,True);
          PopUpShiftKeys;
        end
        else
          DisplayMessage('Invalid KeyName');
        Inc(I);
      end;
    end;
  end;
  Result:=true;
  PopUpShiftKeys;
end;

{AppActivate

This is used to set the current input focus to a given window using its
name.  This is especially useful for ensuring a window is active before
sending it input messages using the SendKeys function.  You can specify
a window's name in its entirety, or only portion of it, beginning from
the left.}

var
  WindowHandle : HWND;

function EnumWindowsProc(WHandle: HWND; lParam: LPARAM): BOOL; export; stdcall;
var
  WindowName : array[0..MAX_PATH] of char;
begin
  {Can't test GetWindowText's return value since some windows don't have a
  title}
  GetWindowText(WHandle,WindowName,MAX_PATH);
  Result := (StrLIComp(WindowName,PChar(lParam), StrLen(PChar(lParam))) <> 0);
```

```
  If (not Result) then
    WindowHandle:=WHandle;
end;

function AppActivate(WindowName : PChar) : boolean;
begin
  try
    Result:=true;
    WindowHandle:=FindWindow(nil,WindowName);
    If (WindowHandle=0) then
      EnumWindows(@EnumWindowsProc,Integer(PChar(WindowName)));
    If (WindowHandle<>0) then
    begin
      SendMessage(WindowHandle, WM_SYSCOMMAND, SC_HOTKEY, WindowHandle);
      SendMessage(WindowHandle, WM_SYSCOMMAND, SC_RESTORE, WindowHandle);
      SetForegroundWindow(WindowHandle);
    end
    else
      Result:=false;
  except
    on Exception do Result:=false;
  end;
end;

end.
```


Author:	Ken Henderson
Contributor:	Irfan Karazor
Added:	2012-07-05
Last updated:	2012-07-05