# How to implement autocompletion in a TEdit

## Answer 1

Here is a procedure using the *OnKeyDown* event that will autocomplete an edit box using a lookup source table. Change it to suit your needs but it should give you an idea of how to do the selections and stuff with an edit control. This will work with just about any type of edit control and I use it for combo boxes as well. You just need to change the typecasting.

```
procedure TForm1.EditKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
var
  s1: string;
  s2: string;
begin
  if TEdit(Sender).Text = '' then
    exit;
  s1 := TEdit(Sender).Text;
  s2 := s1;
  with mtDM.LookTable do  {change here for your own lookup stuff...}
  begin
    if not Locate(LookField, TEdit(Sender).Text,[loPartialKey]) then
    begin
      Key := 0;
      if length(s2) = 1 then
      begin
        TEdit(Sender).Text := '';
        exit;
      end;
      System.delete(s2,length(s2),1);
      TEdit(Sender).Text := s2;
      s1 := s2;
      Locate(LookField, TEdit(Sender).Text,[loPartialKey]);
    end;
    s1 := FieldByName(LookField).AsString;
    TEdit(Sender).Text := Copy(
      s1, 1, length(s2)) + copy(s1, length(s2) + 1, length(s1)
     );
    TEdit(Sender).SelStart := Length(s2);
    TEdit(Sender).SelLength := length(s1) - length(s2);
  end;
  inherited;
end;
```

Tip by Woody

## Answer 2

```
unit AutoEdit;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, StdCtrls, Controls,
  Dialogs, Forms;

type
  TAutoEdit = class(TEdit)
  private
    fList: TListBox;
    fItems: TStringList;
    fLabel: TLabel;
    fCaption: string;
    fBackColor: TColor;
    fCaptionColor: TColor;
    fAutoComplete: Boolean;
    fListCount: Integer;
    fOldText: string;
    procedure SetCaption(S: string);
    procedure SetCaptionColor(const Color: TColor);
    procedure SetBackColor(const Color: TColor);
    procedure SetAutoComplete(AutoCompleteOn: Boolean);
    procedure ShowList;
  protected
    procedure CreateParams( Var params: TCreateParams ); override;
    procedure SetParent(AParent: TWinControl); override;
    procedure SetName(const Value: TComponentName); override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure SetBounds(ALeft, ATop, AWidth, AHeight: Integer); override;
    procedure ListMouseUp(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure HideList;
    procedure DoExit; override;
    property Items: TStringList
      read fItems write fItems;
  published
    procedure KeyPress(var Key: Char); override;
    procedure KeyDown(var Key: Word; Shift: TShiftState); override;
    property Caption: string
      read fCaption write SetCaption;
    property CaptionColor: TColor
      read fCaptionColor write SetCaptionColor;
    property BackColor: TColor
      read fBackColor write SetBackColor;
    property AutoComplete: Boolean
      read fAutoComplete write SetAutoComplete;
    property ListCount: Integer
      read fListCount write fListCount default 5;
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('Freeware', [TAutoEdit]);
end;

{ TAutoEdit }

constructor TAutoEdit.Create(AOwner: TComponent);
begin
  inherited;
  fItems := TStringList.Create;
  fList := TListBox.Create(Self);
  fLabel := TLabel.Create(Self);
  fLabel.ParentColor := True;
  fLabel.AutoSize := False;
  fLabel.FocusControl := Self;
  fCaptionColor := fLabel.Font.Color;
  fBackColor := fLabel.Color;
  fList.Parent := Self;
  fList.IntegralHeight := True;
  fList.ParentCtl3D := False;
  fList.Ctl3D := False;
  fList.TabStop := False;
  fList.Visible := False;
  fListCount := 5;
end;

destructor TAutoEdit.Destroy;
begin
  fItems.Free;
  fLabel.Free;
  inherited;
end;
```

```
procedure TAutoEdit.SetParent(AParent: TWinControl);
var
  FirstSetting: Boolean;
begin
  if Parent = nil then
    FirstSetting := True
  else
    FirstSetting := False;
  inherited;
  if Parent <> nil then
  begin
    fList.Parent := Self.Parent;
    fLabel.Parent := Self.Parent;
    if FirstSetting then
    begin
      fLabel.ParentColor := True;
      SetBounds(Left, Top, Width, Height);
    end;
  end;
end;

procedure TAutoEdit.SetBounds(ALeft, ATop, AWidth, AHeight: Integer);
begin
  inherited SetBounds(ALeft, ATop, AWidth, AHeight);
  if Parent <> nil then
  begin
    if (fCaption > '') and (fLabel.Parent <> nil) then
    begin
      fLabel.Top := ATop - (1 + fLabel.Canvas.TextHeight('lj'));
      fLabel.Height := AHeight + 4 + fLabel.Canvas.TextHeight('lj');
    end
    else
    begin
      fLabel.Top := ATop - 2;
      fLabel.Height := AHeight + 4;
    end;
    fLabel.Left := ALeft - 2;
    fLabel.Width := AWidth + 4;
    if csDesigning in ComponentState then
    begin
      fList.Parent := Self;
      HideList;
    end
    else
    if fList.Visible then
      ShowList;
  end;
end;

procedure TAutoEdit.SetName(const Value: TComponentName);
begin
  if Name > '' then
    if fCaption = Name then
      Caption := Value;
  inherited SetName(Value);
  if Text = Name then
  begin
    Text := '';
    Caption := Value;
  end;
end;

procedure TAutoEdit.CreateParams(var params: TCreateParams);
begin
  inherited;
  fList.Color := Self.Color;
  fList.Font := Self.Font;
  fList.OnMouseUp := ListMouseUp;
  HideList;
end;

procedure TAutoEdit.SetCaption(S: string);
begin
  fCaption := S;
  fLabel.Caption := ' ' + S;
  SetBounds(Left, Top, Width, Height)
end;

procedure TAutoEdit.SetCaptionColor(const Color: TColor);
begin
  if fCaptionColor <> Color then
  begin
    fCaptionColor := Color;
    fLabel.Font.Color := Color;
    SetBounds(Left, Top, Width, Height)
  end;
end;

procedure TAutoEdit.SetBackColor(const Color: TColor);
begin
  if fBackColor <> Color then
  begin
    fBackColor := Color;
    fLabel.Color := Color;
    SetBounds(Left, Top, Width, Height)
  end;
end;

procedure TAutoEdit.SetAutoComplete(AutoCompleteOn: Boolean);
begin
  fAutoComplete := AutoCompleteOn;
end;

procedure TAutoEdit.ListMouseUp(Sender: TObject; Button: TMouseButton;
Shift: TShiftState; X, Y: Integer);
begin
  Text := fList.Items[fList.ItemIndex];
  SelStart := Length(Text);
  HideList;
  fList.Clear;
  PostMessage(Handle, WM_KEYDOWN, VK_TAB, 0);
  PostMessage(Handle, WM_KEYUP, VK_TAB, 0);
end;

procedure TAutoEdit.DoExit;
begin
  if not fList.Focused then
    HideList;
  inherited;
end;

procedure TAutoEdit.KeyPress(var Key: Char);
var
  K, T: string;
  I, S: Integer;
begin
  if ReadOnly then
  begin
    inherited;
    Exit;
  end;
  K := Key;
  if (Key = #27) and (fList.Visible) then
  begin
    Key := #0;
    Text := Copy(Text, 1, SelStart);
    SelStart := Length(Text);
    fList.Clear;
    HideList;
  end
  else
  if fAutoComplete then
    if ((K > #27) and (K < #129)) or (K = #8) then
    begin
      if (K = #8) then
        T := Copy(Text, 1, SelStart - 1)
      else
        T := Copy(Text, 1, SelStart) + K;
      K := Uppercase(T);
      fList.Clear;
      if fItems.Count > 0 then
        for I := 0 to fItems.Count - 1 do
        begin
          if (Pos(K, Uppercase(fItems[I])) = 1) then
            fList.Items.Add(fItems[I]);
          if fList.Items.Count > fListCount - 1 then
            Break;
        end;
      S := Length(T);
      if (fList.Items.Count > 0) and (Key <> #8) then
        Text := Copy(T, 1, S)
          + Copy(fList.Items[0], S + 1, Length(fList.Items[0]))
      else
        Text := T;
      Key := #0;
      SelStart := S;
      SelLength := Length(Text) - S;
      fOldText := Copy(Text, 1, SelStart);
    end;
  if fList.Items.Count > 0 then
    ShowList
  else
    HideList;
  inherited;
end;

procedure TAutoEdit.KeyDown(var Key: Word; Shift: TShiftState);
var
  I, S: Integer;
begin
  if Key = VK_DELETE then
  begin
    fList.Clear;
    HideList;
  end
  else
  if fList.Visible then
    if (Key = VK_DOWN) or (Key = VK_UP) then
    begin
      S := SelStart;
      if Key = VK_DOWN then
        I := fList.ItemIndex + 1
      else
        I := fList.ItemIndex - 1;
      if I < -1 then
        I := fList.Items.Count -1;
      if I > fList.Items.Count - 1 then
        I := - 1;
      fList.ItemIndex := I;
      if I = -1 then
      begin
        Text := fOldText;
        SelStart := Length(Text);
        SelLength := 0;
      end
      else
      begin
        Text := fList.Items[fList.ItemIndex];
        SelStart := S;
        SelLength := Length(Text) - S;
      end;
      Key := 0;
    end;
  if (not fList.Visible) and ((Key = VK_LEFT) or (Key = VK_RIGHT)) then
    if SelLength = Length(Text) then
      if (Shift = []) and (Length(Text) > 0) then
      begin
        SelLength := 0;
        Key := 0;
      end;
  inherited;
end;
```

```
procedure TAutoEdit.ShowList;
begin
  if Parent <> nil then
  begin
    fList.Top := Top + ClientHeight;
    fList.Left := Left;
    fList.Width := Width;
    fList.Height := fList.ItemHeight * (fList.Items.Count + 1);
    fList.BringToFront;
    fList.Show;
  end;
end;

procedure TAutoEdit.HideList;
var
  I: Integer;
begin
  if (Text > '') then
    for I := 0 to fList.Items.Count - 1 do
      if Uppercase(fList.Items[I]) = Uppercase(Text) then
      begin
        Text := fList.Items[I];
        Break;
      end;
  fList.Hide;
  fList.Top := Top;
  fList.Height := 0;
  fList.Left := Left;
  fList.Width := 0;
end;

initialization

RegisterClass(TLabel);

end.
```

Tip by Mike Warren

| | |
|---|---|
| Original resource: | The Delphi Pool |
| Author: | Woody & Mike Warren |
| Added: | 2009-10-26 |
| Last updated: | 2009-10-26 |