









Working with numbers in Edit Boxes

Copyright © 2002 by Horia Tudosie

While the TMaskEdit component will do fine for this job, because of the high number of responses, additional questions and suggestions on this matter in a Delphi forum, I decided to write this small tutorial about creating a new component for solving this goal.

Prerequisites

It is a bad idea to validate by checking keys on the OnKeyPress event. This filters out useful actions like  (copy the value to paste somewhere else),  (Paste),  (used at least to delete all for reentry), , and  (Backspace) etc. It denies legitimate actions considering future actions: you cannot put the dot in another position before deleting the previous dot. It has no control over where the cursor of the mouse has been placed. For floats, there may be also a scientific format...

There is one exception: check in OnKeyPress for . This is for forms that have only one Edit box – OnExit is not fired, because there is nowhere to exit. So check for  to validate. A check can also be done for  to restore the initial or a default value.

If the host form has a TBitButton, OnExit will do. However, if the form has only an edit box and a TSpeedButton, OnExit won't do: TSpeedButton does not take focus - the focus cannot leave the edit box. In this case, validate explicitly from the OnClick of the TSpeedButton.

So where and how to check for valid numbers? Well, in OnExit event. Call the appropriate StrToInt or StrToFloat in a 'try..except' block to check that the string can be converted to a number. Even better, when writing a TNumEdit component put the converted value in a NumValue property of the new component. Better yet, the Val function may return the index of the first unmatched character, and this can help to place the cursor in this position and highlight the rest of the string.

In TNumEdit, is not a good idea to beep – is better to fire an OnWarning event, and let the user programmer to beep or do something else.

TNumEdit

Let's start: Create the TNumEdit component by inheriting from TEdit. Go to the main menu, 'Component/New Component'. See Figure 1.

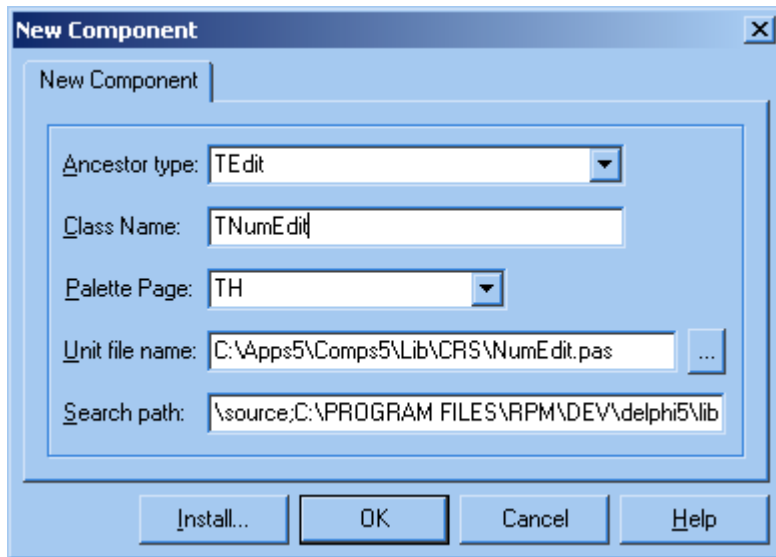


Figure 1 - Create the TNumEdit component

TNumEdit may deal with integer or float values. Define a type to express the kind of number the user wants to use in the NumEdit box, and add the corresponding property:

```

1  unit NumEdit;
2
3  interface
4
5  uses
6    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
7    StdCtrls;
8
9  type
10   TNumEditKind = (neInteger, neFloat);
11
12  type
13   TNumEdit = class(TEdit)
14   private
15     fNumEditKind: TNumEditKind;
16   protected
17   public
18     constructor Create(AOwner: TComponent); override;
19   published
20     property NumberKind: TNumEditKind read fNumEditKind write fNumEditKind
21       default neInteger;
22   end;
23
24  procedure Register;
25
26  implementation
27
28  constructor TNumEdit.Create(AOwner: TComponent);
29  begin
30     inherited;
31     fNumEditKind := neInteger;
32  end;
33
34  procedure Register;
35  begin
36     RegisterComponents('TH', [TNumEdit]);
37  end;
38
39  end.

```

The line 'default neInteger' has to be enforced in the constructor. If the value of NumberKind is not *neInteger*, Delphi does not save it in the runtime type information (RTTI). However, there is no warranty that the component will have by default this value for this property. To enhance this, make it so in the constructor.

Note: How Delphi save the value in RTTI is no relevant for the program and the programmer. Also the default value for fNumEditKind will be the first in the set – neInteger. However is better to add all this in the program for the sake of the code clarity and for making that sure the component will behave the same way on whatever further flavor of Delphi.

Validate

Add a function that validates on request the content of the box. As been said, if this is the only component which can hold the focus on a form (the actions being controlled by TSpeedButtons or popup menus) then you may want to call this function from inside some methods of other components. So, the Validate function stays better it in the public section of TNumEdit:

```
40 Public
41   Function Validate:Boolean;
...
42 function TNumEdit.Validate:boolean;
43 var vi:integer;
44     vf:double;
45     code:integer;
46 begin
47   case NumberKind of
48     neInteger : val(Text,vi,code);
49     neFloat   : val(Text,vf,code);
50   end;
51   result:=code=0;
52   if code<>0 then begin
53     SetFocus;
54     SelStart:=code-1;
55     SelLength:=length(Text);
56   end;
57 end;
```

The Val function from the System unit validates a text that should be converted to a number. The type of validation conforms to the type of its second parameter. The third parameter is an integer that holds 0 if the text represents a valid number or the position of the first character that broke the validation. We can use it to highlight the rest of the string, so the user may just overwrite the error. Keep the focus inside the box, else SelStart and SelLength have no visible effect.

The converted value may be useful in the user program. Obviously, that program deals with numeric values, so let's give it the value. Add a variant property to hold the result.

Is also useful to add an event triggered on bad conversion. This is not quite an error, since focus doesn't leave the box and the user has the chance to correct it, so let's call it OnWarning. In the OnWarning event, the programmer can add code to give instructions about how to fill the box or beep a sound. If the programmer does not fill this even, then let TNumEdit beep an alarm. The System Exclamation Sound seems to be most appropriate.

Another event can be triggered just before the validation so the programmer could change some predefined strings (like 'PI') with their corresponding values.

```

58  private
...
59      fValue      :variant;
60      fOnWarning  :TNotifyEvent;
61      fOnValidate :TNotifyEvent;
...
62  function TNumEdit.Validate:boolean;
63  var vi:integer;
64      vf:double;
65      code:integer;
66  begin
67      if Assigned(OnValidate) then OnValidate(Self);
68      vi:=0; vf:=0;
69      case NumberKind of
70          neInteger : val(Text,vi,code);
71          neFloat   : val(Text,vf,code);
72      end;
73      result:=code=0;
74      if code<>0 then begin
75          SetFocus;
76          SelStart:=code-1;
77          SelLength:=length(Text);
78          if Assigned(OnWarning) then OnWarning(Self)
79          else MessageBeep(MB_ICONEXCLAMATION)
80      end
81      else
82          case NumberKind of
83              neInteger : fValue:=vi;
84              neFloat   : fValue:=vf;
85          end;
86  end;

```

Initialize VI and VF just to avoid a warning from the compiler. Indeed: when code=0 the compiler does not know that the assignments from the second case have valid values.

To test it, create a small project. Place a new TNumEdit component on a form, a TSpeedButton and a TLabel. On the OnClick of the TSpeedButton write:

```

87  procedure TForm1.SpeedButton1Click(Sender: TObject);
88  begin
89      if NumEdit1.Validate then
90          Label1.Caption:=NumEdit1.Value;
91  end;

```

Now is time to use this function when exiting the box. The best is to modify the OnExit event in such a way that the user still can add code for this event without interfering with our checking. Create a new OnExit notify-event and a NewOnExit Procedure to handle it. Modify the default behavior of the component in its constructor:

```

92  published
93      property NumberKind:TNumEditKind read fNumEditKind write fNumEditKind
94          default neInteger;
95      property Value      :variant      read fValue      write SetValue;
96      property OnWarning  :TNotifyEvent read fOnWarning  write fOnWarning;
97      property OnValidate:TNotifyEvent read fOnValidate  write fOnValidate;
98      property OnExit     :TNotifyEvent read fOnExit     write fOnExit;
99  end;

100 constructor TNumEdit.Create(AOwner: TComponent);
101 begin

```

```

102 inherited;
103 fNumEditKind:=neInteger;
104 inherited OnExit:=NewOnExit;
105 end;

106 procedure TNumEdit.NewOnExit(Sender: TObject);
107 begin
108   if Validate then begin
109     if Assigned(fOnExit) then fOnExit(Sender);
110   end
111 end;

```

To test, add another TEdit component and/or a TBitButton in your test project. When moving to the new Edit box or pressing the BitButton, if the NumEdit1 box had a wrong value, NumEdit1 beeps, holds the focus and highlights the wrong characters. However, if the text represents a valid number, the label does not display the value.

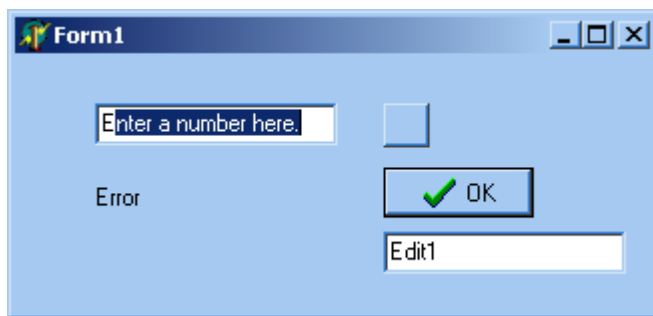


Figure 2 - The main form in the test project.

Add code on the OnExit event of NumEdit1:


```

112 procedure TForm1.NumEdit1Exit(Sender: TObject);
113 begin
114   Label1.Caption:=NumEdit1.Value;
115 end;

```

This could conclude our tutorial, but there are still some aspects that may improve our TNumEdit component.

Improve TNumEdit

The user may want to validate the number without exiting the box. Users usually do that by typing . It may be also useful to restore the previous value. The previous value is the one when the focus enters the component. This value may be a valid number, or just an arbitrary string. The latest is better: the programmer may enter a hint in the box at design time and the user may want to read it again.



So, let's add code for the OnEnter event in the same way we done with OnExit:

```

116 procedure TNumEdit.NewOnEnter(Sender: TObject);
117 begin
118   if Assigned(fOnEnter) then fOnEnter(Sender);
119   IntialStr:=Text;
120 end;




```




Note: is better to save the InitialStr after calling the user's OnEnter event. That gives the user programmer the chance to change the initial value of the box when the user enters it.


To filter the  and  keys just override the KeyPress procedure of the TCustomEdit component from which TNumEdit inherits:

```

121 procedure TNumEdit.KeyPress(var Key: Char);
122 begin
123   if (Key=^M) or (Key=^A) then begin
124     Key:=#0;
125     Validate;
126   end
127   else if Key=^[ then
128     Text:=InitialStr
129   else inherited; // !
130 end;
```

Clear the ^M char with null, so the inherited procedure from TCustomEdit won't beep, but don't do that for  – it is ok to beep for Escape. Moreover, clearing the key does not give the final programmer the chance to process that key again in OnKeyPress, so while there is no choice for , it is for .

When the form contains a default button, ^M does not go to our component – instead, it goes directly to that button. To enhance this feature over that situation, add ^A to the test. ^A corresponds to . Users may remember that where 'Key' does not work, -'Key' does (like  in TMemo!)

Note a small error: the 'else' before 'inherited' will also make  unavailable to the OnKeyPress event of our component; remove 'else' and you get the final code from Figure 3.

```

1  unit NumEdit;
2
3  interface
4
5  uses
6    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
7    StdCtrls;
8
9  type
10   TNumEditKind = (neInteger, neFloat);
11
12  type
13   TNumEdit = class(TEdit)
14   private
15     fNumEditKind : TNumEditKind;
16     fOnExit      : TNotifyEvent;
17     fOnEnter     : TNotifyEvent;
18     fValue       : variant;
19     fOnWarning   : TNotifyEvent;
20     fOnValidate  : TNotifyEvent;
21     InitialStr   : string;
22   protected
23     procedure NewOnExit (Sender: TObject);
24     procedure NewOnEnter(Sender: TObject);
25     procedure SetValue (_Value:Variant);
26     procedure KeyPress(var Key: Char); override;
27   public
28     constructor Create(AOwner: TComponent); override;
29     function Validate:boolean;
30   published
```

```

31     property NumberKind:TNumEditKind read fNumEditKind write fNumEditKind
32         default neInteger;
33     property Value      :variant      read fValue      write SetValue;
34     property OnWarning :TNotifyEvent read fOnWarning  write fOnWarning;
35     property OnValidate:TNotifyEvent read fOnValidate write fOnValidate;
36     property OnExit    :TNotifyEvent read fOnExit    write fOnExit;
37     property OnEnter   :TNotifyEvent read fOnEnter   write fOnEnter;
38 end;
39
40 procedure Register;
41
42 implementation
43
44 constructor TNumEdit.Create(AOwner: TComponent);
45 begin
46     inherited;
47     fNumEditKind:=neInteger;
48     inherited OnExit :=NewOnExit;
49     inherited OnEnter:=NewOnEnter;
50 end;
51
52 procedure TNumEdit.NewOnExit(Sender: TObject);
53 begin
54     if Validate then begin
55         if Assigned(fOnExit) then fOnExit(Sender);
56     end
57 end;
58
59 procedure TNumEdit.NewOnEnter(Sender: TObject);
60 begin
61     if Assigned(fOnEnter) then fOnEnter(Sender);
62     InitialStr:=Text;
63 end;
64
65 function TNumEdit.Validate:boolean;
66 var vi:integer;
67     vf:double;
68     code:integer;
69 begin
70     if Assigned(OnValidate) then OnValidate(Self);
71     vi:=0; vf:=0;
72     case NumberKind of
73         neInteger : val(Text,vi,code);
74         neFloat   : val(Text,vf,code);
75     end;
76     result:=code=0;
77     if code<>0 then begin
78         SetFocus;
79         SelStart:=code-1;
80         SelLength:=length(Text);
81         if Assigned(OnWarning) then OnWarning(Self)
82         else Messagebeep(MB_ICONEXCLAMATION)
83     end
84     else
85         case NumberKind of
86             neInteger : fValue:=vi;
87             neFloat   : fValue:=vf;
88         end;
89 end;
90
91 procedure TNumEdit.SetValue(_Value:Variant);
92 begin //dummy - Read only property
93 end;
94
95 procedure TNumEdit.KeyPress(var Key: Char);
96 begin
97     if (Key=^M) or (Key=#10) then begin
98         Key:=#0;
99         Validate;
100    end
101    else if Key=^[ then

```

```





102   Text:=InitialStr;
103   inherited;
104 end;
105
106 procedure Register;
107 begin
108   RegisterComponents('TH', [TNumEdit]);
109 end;
110
111 end.

```

Figure 3 – TNumEdit Code.

The Test Program

The listing from Figure 4 and the form from Figure 2 are given as templates. Experiment with them by adding or removing various routines on the corresponding events of the TNumEdit component. Also switch the Default property of the BitButton1.

NumEdit1KeyPress process the keys  (Backspace),  and  by replacing the content of the NumEdit1 box with the strings ‘’ (deletes all), ‘pi’ and ‘e’, while the NumEdit1Validate procedure replaces those strings with their corresponding values just before the validation – when the user exits the NumEdit1 box or types .

Note the definition of CtrlBackspace in line 66: a good listing may not need too many comments in Pascal!

```

1  unit Unit1;
2
3  interface
4
5  uses
6    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
7    StdCtrls, NumEdit, Buttons;
8
9  type
10   TForm1 = class(TForm)
11     NumEdit1: TNumEdit;
12     SpeedButton1: TSpeedButton;
13     Label1: TLabel;
14     Edit1: TEdit;
15     BitBtn1: TBitBtn;
16     procedure SpeedButton1Click(Sender: TObject);
17     procedure NumEdit1Exit(Sender: TObject);
18     procedure NumEdit1Warning(Sender: TObject);
19     procedure NumEdit1Enter(Sender: TObject);
20     procedure NumEdit1Validate(Sender: TObject);
21     procedure NumEdit1KeyPress(Sender: TObject; var Key: Char);
22   private
23     { Private declarations }
24   public
25     { Public declarations }
26   end;
27
28   var
29     Form1: TForm1;
30
31   implementation
32
33     {$R *.DFM}
34
35   procedure TForm1.SpeedButton1Click(Sender: TObject);
36   begin
37     if NumEdit1.Validate then
38       Label1.Caption:=NumEdit1.Value;
39   end;
40

```

```

41 procedure TForm1.NumEdit1Exit(Sender: TObject);
42 begin
43     Label1.Caption:=NumEdit1.Value;
44 end;
45
46 procedure TForm1.NumEdit1Warning(Sender: TObject);
47 begin
48     Label1.Caption:='Error';
49     //Messagebeep(MB_ICONEXCLAMATION);
50 end;
51
52 procedure TForm1.NumEdit1Enter(Sender: TObject);
53 begin
54     NumEdit1.Text:='Enter a number here.';
55 end;
56
57 procedure TForm1.NumEdit1Validate(Sender: TObject);
58 begin
59     if UpperCase(Trim(NumEdit1.Text))='PI' then
60         NumEdit1.Text:='3.141592653589793'
61     else if UpperCase(Trim(NumEdit1.Text))='E' then
62         NumEdit1.Text:=FloatToStr(Exp(1));
63 end;
64
65 procedure TForm1.NumEdit1KeyPress(Sender: TObject; var Key: Char);
66 const CtrlBackspace=#127;
67 begin
68     case Key of
69         CtrlBackspace : begin NumEdit1.Text:=''; key:=#0; end;
70         ^P             : begin NumEdit1.Text:='pi'; key:=#0; end;
71         ^E             : begin NumEdit1.Text:='e'; key:=#0; end;
72         else inherited;
73     end;
74 end;
75
76 end.

```

Figure 4 – Test program.

Conclusion

This tutorial shows how to create a new component that validates and converts the keyed data in a box to either an integer or a float number. It shows step by step how to override the inherited OnEnter and OnExit events when the parent class does not have internal procedures to override, and how to change the OnKeyPress event by overrating the private KeyPress procedure of the parent. The component exposes the converted value in a new property as a variant to hold both integer and float values. The tutorial also covers some aspects of how to use it on a form together with some other components that can or cannot hold the focus. New events were added to provide more functionality. No previous functionality has been compromised.