Exploring the rich & versatile inplace editing capabilities of TAdvStringGrid

Introduction

As soon as you want to use a grid for more than just presenting information, the availability of strong inplace editing capabilities is crucial for creating user-friendly applications. In TAdvStringGrid, not only a very wide range of built-in inplace editors is available but there is also an extensive and fine grained control over when inplace editors appear and how they interact with the various key events in the grid. TAdvStringGrid was designed to enable most scenarios users want with just setting a few properties rather than writing a lot of code. In this article, we start with a brief introduction of the basic inplace editors. Then a more detailed look is given to inplace editing and navigation. Next, more complex built-in inplace editors are handled and finally, there is a section on using any TWinControl descendent control as inplace editor in the grid.

Basic editing capabilities in the grid

First of all, editing in the grid is enabled by setting goEditing to true in grid.Options. In code, this can be done with:

Delphi
```
grid.Options := grid.Options + [goEditing];
```

C++
```
AdvStringGrid1->Options = AdvStringGrid1->Options << goEditing;
```

This enables editing for all cells in the grid except fixed cells. In many cases it is desirable that editing is enabled only in some specific cells or columns.
A cell can be set as readonly by using: grid.ReadOnly[column,row]: boolean and setting this property to true. An alternative is by doing this dynamically via the event OnCanEdit.
In this code snippet, editing is only possible in columns 2 and 4:

Delphi:
```
procedure TForm1.AdvStringGrid1CanEditCell(Sender: TObject; ARow, ACol: Integer;
  var CanEdit: Boolean);
begin
  CanEdit := ACol in [2,4]
end;
```

C++:
```
void __fastcall TForm1::AdvStringGrid1CanEditCell(TObject *Sender, int ARow, int ACol,
                bool &CanEdit)
{
  CanEdit = (ACol == 2)       | (ACol == 4);
}
```

Typically, editing is started by:
1) a mouse click on the focused cell
2) pressing F2
3) typing any character

A few settings that allow control over this default behaviour are:
grid.MouseActions.DirectEdit: boolean;

When true, the inplace editor is shown after the first mouseclick

grid.MouseActions.EditOnDblClickOnly: boolean;

When true, the inplace editor is only shown after a double click

grid.MouseActions.CaretPositioning: boolean;

When false, the inplace editor starts with all text selected and the caret after the last character. When true, the caret is positioned at the location where the mouse was clicked to start the inplace editing.

grid.MouseActions.EditSelectAll: boolean;

When false, the caret is positioned after the last character but no text is selected, allowing to immediately type any characters without overwriting the selection.

Editing and navigation in the grid

Often it is desirable to make it convenient and fast to fully operate the editing in the grid with as few keypresses as possible. Therefore it is often convenient to automatically start the inplace editor for the next sequential cell when pressing the return key. To enable this capability, set grid.Navigations.AdvanceOnEnter := true. Typically, the next sequential cell is the cell right from the current cell, ie. Editing sequence is from left to right. In some cases, it can be required that the editing sequence is from top cell to bottom cell. The direction can be choosen with the property grid.Navigation.AdvanceDirection. Similary, it can be convenient that the inplace editor is also immediately shown when the user moves to a next cell with the arrow keys. To have this behaviour, set grid.Navigation.AlwaysEdit to true. When you want to allow that the user uses the TAB key to move to the next cell, set goTabs to true in grid.Options. Normally, the TAB key moves focus between controls. With goTabs set to true, TAB key moves focus in cells of the grid and by default, when the last cell (bottom right cell) is reached, the TAB key moves focus back to the first cell. If you want that the focus moves to the next control on the form when TAB is pressed on the last cell, set grid.Navigation.TabToNextAtEnd = true. Another interesting feature is called CursorWalkEditor. When grid.Navigation.CursorWalkEditor is set to true, the left & right arrow keys will move the focus to the previous or next cell when either the LEFT Key is pressed when the caret is at position 0 in the editor or when the RIGHT key is pressed when the caret is after the last character.

Built-in regular editors and using their properties

The default editor is the type edNormal. It is set with grid.DefaultEditor. This is the default editor type that is used for all cells in the grid. The edNormal inplace editor type is similar to a regular TEdit control. Any character is allowed, there is no size limitation and multiline text can be entered by using CTRL-ENTER to start a new line.

Fine-tuning the basic edNormal editor:
- When multi-line text should not be allowed, set grid.Navigation.AllowCtrlEnter = false
- When the length of the entered text should be limited, set grid.MaxEditLength to a value larger than zero. When MaxEditLength is set to zero, it is ignored.

Variations of this basic editor type are:
edNumeric : allow numbers only with a sign
edPositiveNumeric : allow positive numbers only
edFloat: allow a floating point number, ie. number, decimal separator and thousand separator
edUpperCase: added characters are forced to upper-case.
edMixedCase: added characters are forced to mixed-case, ie. auto capitalizing the first letter of words.
edLowerCase: added characters are forced to lower-case.
edValidChars: allow only the characters that are in the set grid.ValidChars: set of char;
edEditBtn: edit control with embedded button. A click on this embedded button triggers the OnEllipsClick event.
edNumericEditBtn: numeric edit control with embedded button
edFloatEditBtn: float edit control with embedded button

This sample code snippet selects different editor types for different columns.

Delphi:
```
procedure TForm1.AdvStringGrid1GetEditorType(Sender: TObject; ACol,
  ARow: Integer; var AEditor: TEditorType);
```

```
begin
  AdvStringGrid1.MaxEditLength := 255;
  case ACol of
  1:
    begin
      AEditor := edNumeric;
      AdvStringGrid1.MaxEditLength := 4;
    end;
  2:
    begin
      AEditor := edMixedCase;
    end;
  3:
    begin
      AEditor := edValidChars;
      AdvStringGrid1.ValidChars := 'ABCDEF0123456789';
      AdvStringGrid1.MaxEditLength := 8;
    end;
  end;
end;
```

C++:

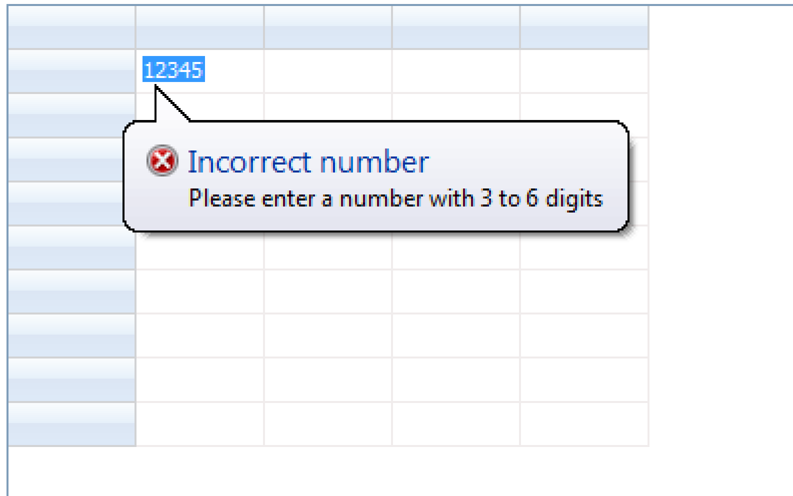
```
void __fastcall TForm1::AdvStringGrid1GetEditorType(TObject *Sender, int ACol, int ARow,
                 TEditorType &AEditor)
{
  AdvStringGrid1->MaxEditLength = 255;
  switch (ACol) {
  case 1:
        {
                AEditor = edNumeric;
                AdvStringGrid1->MaxEditLength = 4;
                break;
        }
  case 2:
        {
                AEditor = edMixedCase;
                break;
        }
  case 3:
        {
                AEditor = edValidChars;
                AdvStringGrid1->ValidChars = "ABCDEF0123456789";
                AdvStringGrid1->MaxEditLength = 8;
                break;
        }
  }
}
```

In this code snippet, the editor in the first column accepts only a numeric value with maximum 4 digits. In the second column, a mixed case editor is specified and in the last column an 8 digit hexadecimal value only is allowed.

Validation after editing

While the grid has several inplace editor types that automatically restrict entry thereby disallowing users to enter unwanted or incorrect data, this is not always possible. Therefore, in many cases, validation is required when the user is about to stop editing. In TAdvStringGrid, the event OnCellValidate is triggered with as parameters the coordinates of the cell being edited, the new value that is about to be entered and a parameter to indicate this value is valid or not. When this Valid parameter is set to false, inplace editing is not stopped, forcing the user to enter a valid value. As the Value parameter is also a variable parameter, it can also be used for auto correcting purposes. In this sample code snippet, the user should enter a numeric value between 3 and 6 digits and when valid, the value is auto corrected to have a dollar sign suffix. In addition, a balloon is used to inform what exactly is incorrect. The grid has public properties:

grid.InvalidEntryTitle: string; Title of the balloon
grid.InvalidEntryText: string; Text of the balloon
grid.InvalidEntryIcon: integer; Icon of the balloon



Delphi:
```
procedure TForm1.AdvStringGrid1CellValidate(Sender: TObject; ACol,
  ARow: Integer; var Value: string; var Valid: Boolean);
var
  len: integer;
begin
  len := Length(Value);
  Valid := (len >= 3) and (len <= 6);
  if Valid then
    Value := Value + '$'
  else
  begin
    AdvStringGrid1.InvalidEntryTitle := 'Incorrect number';
    AdvStringGrid1.InvalidEntryText := 'Please enter a number with 3 to 6 digits';
    AdvStringGrid1.InvalidEntryIcon := ieError;
  end;
end;

procedure TForm1.AdvStringGrid1GetEditorType(Sender: TObject; ACol,
  ARow: Integer; var AEditor: TEditorType);
begin
  AEditor := edNumeric;
end;
```

C++:
```
void __fastcall TForm2::AdvStringGrid1CellValidate(TObject *Sender, int ACol, int ARow,
                 UnicodeString &Value, bool &Valid)
{
  int len = Value.Length();
  Valid = (len >= 3) & (len <= 6);
  if (Valid)
        Value = Value + '$';
  else
  {
        AdvStringGrid1->InvalidEntryTitle = "Incorrect number";
        AdvStringGrid1->InvalidEntryText = "Please enter a number with 3 to 6 digits";
        AdvStringGrid1->InvalidEntryIcon = ieError;
  }

}
```

```
void __fastcall TForm2::AdvStringGrid1GetEditorType(TObject *Sender, int ACol, int ARow,
                TEditorType &AEditor)
{
  AEditor = edNumeric;
}
```

Combobox and spin editors

Another type of inplace editors are comboboxes and spin editors. The types defined are:
edComboEdit: editable combobox (csDropDown style)
edComboList: non editable combobox (csDropDownList style)
edSpinEdit: numeric spin editor
edFloatSpinEdit: floating point spin editor

For the comboboxes, values can be accessed with grid.ComboBox.Items or also with methods grid.ClearComboString, grid.AddComboString, grid.AddComboStringObject. The editor type is also set from the OnGetEditorType event and the values for the combobox can be set from the event OnGetEditorProp. The value of the combobox can also be preset with grid.SetComboSelection(ItemIndex) or grid.SetComboSelectionString(string). To make it clear how this works, this sample shows the use of two different comboboxes and two different spin editors:

Delphi:

```
procedure TForm1.AdvStringGrid1GetEditorProp(Sender: TObject; ACol,
  ARow: Integer; AEditLink: TEditLink);
var
  i: integer;
begin
  case ACol of
  1:
    begin
      AdvStringGrid1.ClearComboString;
      for i := 0 to 10 do AdvStringGrid1.AddComboStringObject(IntToStr(i),TObject(i));
    end;
  2:
    begin
      AdvStringGrid1.ClearComboString;
      AdvStringGrid1.AddComboString('BMW');
      AdvStringGrid1.AddComboString('Audi');
      AdvStringGrid1.AddComboString('Porsche');
      AdvStringGrid1.AddComboString('Ferrari');
      AdvStringGrid1.AddComboString('Mercedes');
      // preset the selection to Mercedes
      AdvStringGrid1.SetComboSelection(4);
    end;
  3:
    begin
      AdvStringGrid1.SpinEdit.MinValue := 0;
      AdvStringGrid1.SpinEdit.MaxValue := 100;
      AdvStringGrid1.SpinEdit.Increment := 1;
    end;
  4:
    begin
      AdvStringGrid1.SpinEdit.MinFloatValue := -1.5;
      AdvStringGrid1.SpinEdit.MaxFloatValue := +1.5;
      AdvStringGrid1.SpinEdit.IncrementFloat := 0.01;
    end;
  end;

end;

procedure TForm1.AdvStringGrid1GetEditorType(Sender: TObject; ACol,
  ARow: Integer; var AEditor: TEditorType);
begin
```

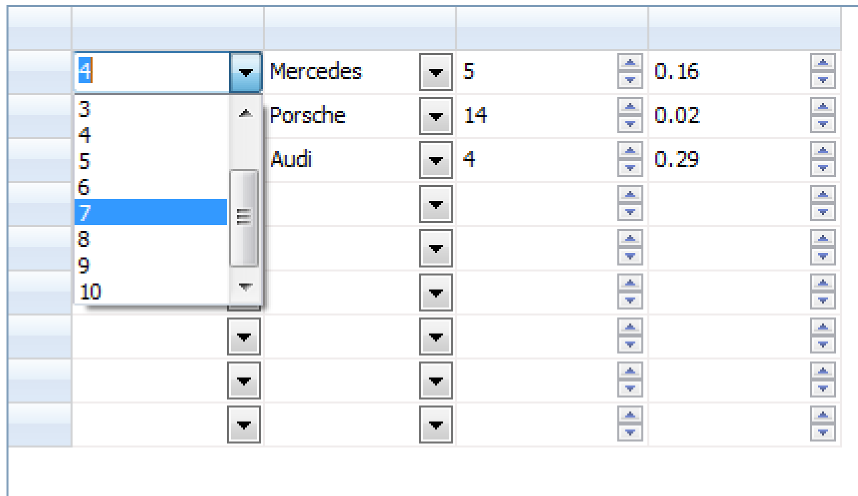
```
  case ACol of
  1: AEditor := edComboEdit;
  2: AEditor := edComboList;
  3: AEditor := edSpinEdit;
  4: AEditor := edFloatSpinEdit;
  end;
end;
```

C++:

```
void __fastcall TForm2::AdvStringGrid1GetEditorType(TObject *Sender, int ACol, int ARow,

                 TEditorType &AEditor)
{
  switch (ACol) {
  case 1:
       AEditor = edComboEdit;
       break;
  case 2:
       AEditor = edComboList;
       break;
  case 3:
       AEditor = edSpinEdit;
       break;
  case 4:
       AEditor = edFloatSpinEdit;
       break;
  }
}

void __fastcall TForm2::AdvStringGrid1GetEditorProp(TObject *Sender, int ACol, int ARow,
                 TEditLink *AEditLink)
{
  int i;
  switch (ACol)
  {
  case 1:
         AdvStringGrid1->ClearComboString();
         for (i=0; i <10; i++)
         {
              AdvStringGrid1->AddComboStringObject(IntToStr(i),reinterpret_cast<TObject
*>(i));
         }
         break;
  case 2:
         AdvStringGrid1->ClearComboString();
         AdvStringGrid1->AddComboString("BMW");
         AdvStringGrid1->AddComboString("Audi");
         AdvStringGrid1->AddComboString("Porsche");
         AdvStringGrid1->AddComboString("Ferrari");
         AdvStringGrid1->AddComboString("Mercedes");
          // preset the selection to Mercedes
         AdvStringGrid1->SetComboSelection(4);
         break;
  case 3:
         AdvStringGrid1->SpinEdit->MinValue = 0;
         AdvStringGrid1->SpinEdit->MaxValue = 100;
         AdvStringGrid1->SpinEdit->Increment = 1;
         break;
  case 4:
         AdvStringGrid1->SpinEdit->MinFloatValue = -1.5;
         AdvStringGrid1->SpinEdit->MaxFloatValue = +1.5;
         AdvStringGrid1->SpinEdit->IncrementFloat = 0.01;
         break;
  }
}
```

Notice that by default, the combobox or spin editor only appears when the editing starts. It can be desirable to have the combobox or spin editor always visible so that the user is aware that these cells do not have a regular editor. This can be enabled by setting:

grid.ControlLook.DropDownAlwaysVisible := true
grid.ControlLook.SpinButtonsAlwaysVisible := true



Date & time inplace editors

For editing time, date or combined time and date values in the grid, different editors are available:

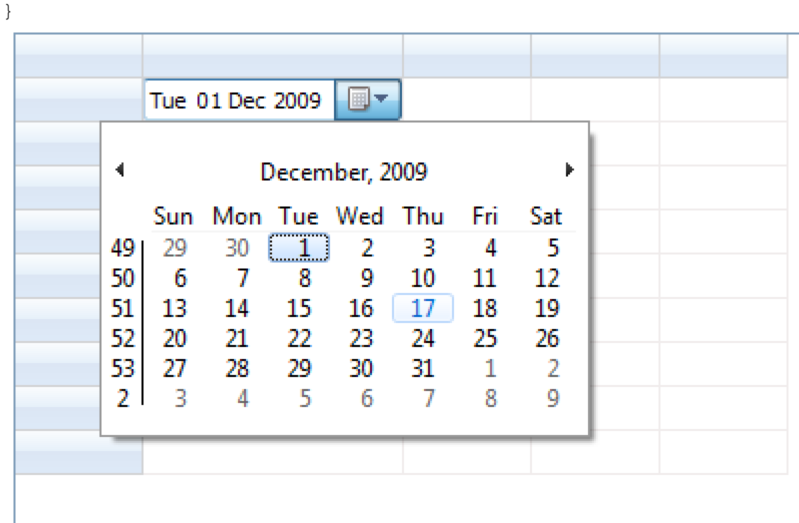
edDateEdit : Windows datepicker control with dropdown calendar
edTimeEdit : Windows timepicker control
edDateEditUpDown: Windows datepicker with spin editor
edDateSpinEdit: VCL date spin editor
edTimeSpinEdit: VCL time spin editor
edDateTimeEdit: combined date and time editor

The edDateEdit, edTimeEdit inplace editor can also be directly accessed via grid.DateTimePicker to further fine-tune properties such as formatting of date/time display, appearance of the calendar etc..
To demonstrate this, the code below shows how the format of the date can be controlled for the date picker and weeknumbers are turned on on the calendar while the default display of today's date is disabled:

Delphi:
```
procedure TForm2.AdvStringGrid1GetEditorType(Sender: TObject; ACol,
  ARow: Integer; var AEditor: TEditorType);
begin
  AEditor := edDateEdit;
  AdvStringGrid1.DateTimePicker.Weeknumbers := true;
  AdvStringGrid1.DateTimePicker.ShowToday := false;
  AdvStringGrid1.DateTimePicker.Format := 'ddd dd MMM yyyy';
end;
```

C++:
```
void __fastcall TForm2::AdvStringGrid1GetEditorType(TObject *Sender, int ACol, int ARow,
                 TEditorType &AEditor)
{
  AEditor = edDateEdit;
  AdvStringGrid1->DateTimePicker->Weeknumbers = true;
  AdvStringGrid1->DateTimePicker->ShowToday = false;
  AdvStringGrid1->DateTimePicker->Format = "ddd dd MMM yyyy";
```

```
}
```



Dropdown editors

For an even more rich user experience, TAdvStringGrid v5 introduces a new set of inplace editors for choosing colors, images, time, edit numbers via a calculator, pick values from a combobox with detail notes per item or pick values from a dropdown grid. This set of inplace editors shares a common structure. The dropdown has a header and footer. Both header and footer can contain HTML formatted informational text about the editor and can feature buttons as well. The settings for the dropdown control header and footer are exposed via grid.ControlLook.DropDownHeader and grid.ControlLook.DropDownFooter. Note that the dropdown header and footer are optional and can be turned off by setting the respective Visible property to false. When the SizeGrid property is set to true on the footer, the dropdown can be resized by dragging from the bottom-right corner. Using the time picker, memo, trackbar and calculator dropdown is straightforward. Just like with all other edit controls, use the OnGetEditorType event and set the editor to the correct editor type. For the color picker and image picker, some more detailed interaction with the grid is available. By default, the color picker will set the cell color to the color choosen and will trigger the event OnColorSelected. If we have added a shape in the cell though, it is just the color of the shape that the color picker will set. To demonstrate this, add following code:

Delphi:

```
procedure TForm2.AdvStringGrid1GetEditorType(Sender: TObject; ACol,
  ARow: Integer; var AEditor: TEditorType);
begin
  // set this editor just for cell 1,1
  if (Acol = 1) and (ARow = 1) then
  begin
    AEditor := edColorPickerDropDown;
    // select the colorcube as color selector
    AdvStringGrid1.ColorPickerDropDown.ColorSelectionStyle := csColorCube;
  end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
  AdvStringGrid1.Options := AdvStringGrid1.Options + [goEditing];
  AdvStringGrid1.AddShape(1,1,csRectangle, clWhite, clBlack, haBeforeText, vaCenter);
end;
```

C++:

```
void __fastcall TForm2::AdvStringGrid1GetEditorType(TObject *Sender, int ACol, int ARow,
                 TEditorType &AEditor)
{
  // set this editor just for cell 1,1
  if (Acol == 1) && (ARow == 1)
  {
```

```
        AEditor = edColorPickerDropDown;
        // select the colorcube as color selector
        AdvStringGrid1->ColorPickerDropDown->ColorSelectionStyle = csColorCube;
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm2::FormCreate(TObject *Sender)
{
  AdvStringGrid1->AddShape(1,1,csRectangle,clWhite, clBlack, haBeforeText, vaCenter);
  AdvStringGrid1->Options = AdvStringGrid1->Options << goEditing;
}
```

Similar to a color picker, an image picker dropdown can also be used to edit an imagelist image set in a cell. By default, it will just trigger the OnImageSelected event when editing is done, but when a cell has an imagelist image, it will also automatically update this image. Again, with very little code this can be achieved. Drop an ImageList on the form and assign it to grid.GridImages and add the code:

Delphi:

```
procedure TForm2.AdvStringGrid1GetEditorType(Sender: TObject; ACol,
  ARow: Integer; var AEditor: TEditorType);
begin
  if (Acol = 1) and (ARow = 1) then
  begin
    AEditor := edImagePickerDropDown;
    // will automatically load all images from the imagelist in the image picker
    AdvStringGrid1.ImagePickerDropDown.AddImagesFromImageList;
    // forces the imagepicker to display images in 2 columns
    AdvStringGrid1.ImagePickerDropDown.Columns := 2;
  end;
end;

procedure TForm2.FormCreate(Sender: TObject);
begin
  AdvStringGrid1.Options := AdvStringGrid1.Options + [goEditing];
  AdvStringGrid1.AddDataImage(1,1,0,haBeforeText, vaCenter);
end;
```

C++:

```
void __fastcall TForm2::AdvStringGrid1GetEditorType(TObject *Sender, int ACol, int ARow,
                TEditorType &AEditor)
{
  // set this editor just for cell 1,1
  if (Acol == 1) && (ARow == 1)
  {
        AEditor = edImagePickerDropDown;
        // will automatically load all images from the imagelist in the image picker
        AdvStringGrid1->ImagePickerDropDown->AddImagesFromImageList();
        // forces the imagepicker to display images in 2 columns
        AdvStringGrid1->ImagePickerDropDown->Columns = 2;
  }
}

void __fastcall TForm2::FormCreate(TObject *Sender)
{
  AdvStringGrid1->AddDataImage(1,1,0,haBeforeText, vaCenter),
  AdvStringGrid1->Options = AdvStringGrid1->Options << goEditing;
}
```

The detail picker dropdown can be considered as a combobox with an optional extra image per item and additional notes text for each item. Its use is straightforward and becomes clear with following code:

Delphi:

```
procedure TForm2.AdvStringGrid1GetEditorType(Sender: TObject; ACol,
  ARow: Integer; var AEditor: TEditorType);
```

```
begin
  AEditor := edDetailDropDown;
end;

procedure TForm2.FormCreate(Sender: TObject);
var
  di: TDetailItem;
begin
  AdvStringGrid1.DetailPickerDropDown.Images := ImageList1;
  AdvStringGrid1.DetailPickerDropDown.ItemHeight := 40;

  AdvStringGrid1.DetailPickerDropDown.Items.Clear;
  di := AdvStringGrid1.DetailPickerDropDown.Items.Add;

  di.ImageIndex := 0;
  di.Caption := 'Delphi';
  di.Notes := 'Most productive IDE for Win32 development';

  di := AdvStringGrid1.DetailPickerDropDown.Items.Add;
  di.ImageIndex := 1;
  di.Caption := 'Delphi Prism';
  di.Notes := 'Take your Pascal skills to .NET';

  di := AdvStringGrid1.DetailPickerDropDown.Items.Add;
  di.ImageIndex := 2;
  di.Caption := 'Delphi PHP';
  di.Notes := 'RAD development for PHP';
end;
```

C++:

```
void __fastcall TForm2::AdvStringGrid1GetEditorType(TObject *Sender, int ACol, int ARow,
                 TEditorType &AEditor)
{
  AEditor = edDetailDropDown;
}

void __fastcall TForm2::FormCreate(TObject *Sender)
{
  TDetailItem *di;

  AdvStringGrid1->DetailPickerDropDown->Images = ImageList1;
  AdvStringGrid1->DetailPickerDropDown->ItemHeight = 40;

  AdvStringGrid1->DetailPickerDropDown->Items->Clear;
  di = AdvStringGrid1->DetailPickerDropDown->Items->Add();

  di->ImageIndex = 0;
  di->Caption = "Delphi";
  di->Notes = "Most productive IDE for Win32 development";

  di = AdvStringGrid1->DetailPickerDropDown->Items->Add();
  di->ImageIndex = 1;
  di->Caption = "Delphi Prism";
  di->Notes = "Take your Pascal skills to ->NET";

  di = AdvStringGrid1->DetailPickerDropDown->Items->Add();
  di->ImageIndex = 2;
  di->Caption = "Delphi PHP";
  di->Notes = "RAD development for PHP";
}
```

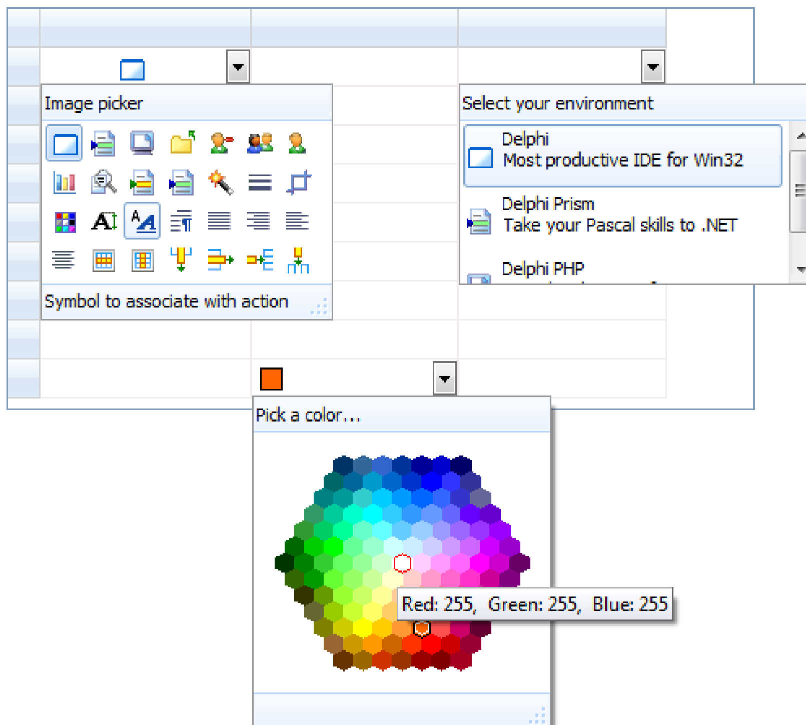
Finally it is possible to have a grid as inplace editor. The value that will be displayed in the cell is the value from the column in the grid on the selected row that is set as lookup column with property GridDropdown.LookupColumn. To set the properties for each column in the grid, the grid.Columns collection is

available. Via this column of type TDropDownColumn, it can be defined whether a column contains text or an imagelist image. The items in the grid can be added via grid.Items which is a collection of TDropDownItem objects. How everything falls into place is made clear with the sample code to initialize a dropdown grid:

Delphi:
```
var
  dc: TDropDownColumn;
  di: TDropDownItem;
begin
  AdvStringGrid1.GridDropDown.Images := ImageList1;
  dc := AdvStringGrid1.GridDropDown.Columns.Add;
  dc.Header := '';
  dc.ColumnType := ctImage;
  dc.Width := 30;
  dc := AdvStringGrid1.GridDropDown.Columns.Add;
  dc.Header := 'Brand';
  dc.ColumnType := ctText;
  dc := AdvStringGrid1.GridDropDown.Columns.Add;
  dc.Header := 'Type';
  dc.ColumnType := ctText;

  di := AdvStringGrid1.GridDropDown.Items.Add;
  di.ImageIndex := 0;
  di.Text.Add('');
  di.Text.Add('BMW');
  di.Text.Add('7 series');

  di := AdvStringGrid1.GridDropDown.Items.Add;
  di.ImageIndex := 1;
  di.Text.Add('');
  di.Text.Add('Mercedes');
  di.Text.Add('S class');

  di := AdvStringGrid1.GridDropDown.Items.Add;
  di.ImageIndex := 2;
  di.Text.Add('');
  di.Text.Add('Porsche');
  di.Text.Add('911');

  di := AdvStringGrid1.GridDropDown.Items.Add;
  di.ImageIndex := 3;
  di.Text.Add('');
  di.Text.Add('Audi');
  di.Text.Add('A8');

  AdvStringGrid1.GridDropDown.LookupColumn := 1;
end;
```

C++:
```
{
 TDropDownColumn *dc;
 TDropDownItem *di;

 AdvStringGrid1->GridDropDown->Images = ImageList1;
 dc = AdvStringGrid1->GridDropDown->Columns->Add();
 dc->Header = "";
 dc->ColumnType = ctImage;
 dc->Width = 30;
 dc = AdvStringGrid1->GridDropDown->Columns->Add();
 dc->Header = "Brand";
 dc->ColumnType = ctText;
 dc = AdvStringGrid1->GridDropDown->Columns->Add();
 dc->Header = "Type";
```

```
 dc->ColumnType = ctText;

 di = AdvStringGrid1->GridDropDown->Items->Add();
 di->ImageIndex = 0;
 di->Text->Add("");
 di->Text->Add("BMW");
 di->Text->Add("7 series");

 di = AdvStringGrid1->GridDropDown->Items->Add();
 di->ImageIndex = 1;
 di->Text->Add("");
 di->Text->Add("Mercedes");
 di->Text->Add("S class");

 di = AdvStringGrid1->GridDropDown->Items->Add();
 di->ImageIndex = 2;
 di->Text->Add("");
 di->Text->Add("Porsche");
 di->Text->Add("911");

 di = AdvStringGrid1->GridDropDown->Items->Add();
 di->ImageIndex = 3;
 di->Text->Add("");
 di->Text->Add("Audi");
 di->Text->Add("A8");

 AdvStringGrid1->GridDropDown->LookupColumn = 1;
}
```



Using custom editors

Finally, if the wide range of built-in editors is not sufficient for your needs, the grid offers the capability to use any TWinControl descendent control as inplace editor for the grid. There are basically two ways to do this. First way is to create a class descending from TEditLink that implements the interface between your edit control and the grid. Implementing this class allows fine grained control how the editor should interact with the grid. Describing this in detail deserves an article on itself. You can find more information at http://www.tmssoftware.com/site/asg24.asp . The second way is a lot faster. Drop a TFormControlEditLink on the form and the edit control you want to use as inplace editor. Assign the control to TFormControlEditLink.Control. Implement the grid's OnGetEditorType event as:

```
procedure TForm1.AdvStringGrid1GetEditorType(Sender: TObject; ACol,
  ARow: Integer; var AEditor: TEditorType);
begin
  case ACol of
  1:
    begin
      AEditor := edCustom;
      AdvStringGrid1.EditLink := FormControlEditLink1;
    end;
  end;
end;
```

From now, starting editing in the grid will show the control as inplace editor and leaving focus hides this inplace editor. Only thing left is to implement two events for TFormControlEditLink that will transfer the value of the control to the grid and vice versa. In this example, this is achieved with:

```
procedure TForm1.FormControlEditLink1GetEditorValue(Sender: TObject;
  Grid: TAdvStringGrid; var AValue: string);
begin
  AValue := PlannerDatePicker1.Text;
end;

procedure TForm1.FormControlEditLink1SetEditorValue(Sender: TObject;
  Grid: TAdvStringGrid; AValue: string);
begin
  PlannerDatePicker1.Text := AValue;
end;
```