

## Editing complex Data Types with the Telerik WPF RadGridView

Technorati-Tags: [telerik](#), [RFID](#), [WPF](#)

In my [RFID Blog Posts](#) I promised to tell something about “List controls” holding my custom controls.

Here it is – but with a little change – instead of the (really simple) approach to use custom controls I decided to extend the [telerik](#) WPF [RadGridView](#) with a custom editor.

Unfortunately a closer description about this feature is not available in the documentation. So I decide to investigate those things and provide a blog post (this one) about it.

What you find in the documentation sounds really good:

*The GridView supports several types of editors - **CheckBoxEditor**, **ComboBoxEditor**, **DatePickerEditor** and **TextBoxEditor**. You can either explicitly specify (explained further in this topic) what editor should be used with a column or you can use the automatic detection of the editor that is appropriate according to the **DataType** of the column. The automatic detection functionality is built-in and you don't need any code to use it.*

Following this I assumed that the RadGridView is able to choose the correct built-in editor. BUT – the control can do much more – it can also assign custom editor automatically. The first thought may be: ...lots of interface, a huge number of “integration code” and so on.

Nothing – last not least it's nothing more than 3 Lines of code 😊

OK – it is a bit more if you like to have full functionality – but if basic functionality is enough it is:

- 1.) One line to assign the custom editor to the RadGridView
- 2.) One line (an empty class definition) for a support class
- 3.) One line (also an empty class definition) for the editor itself

And the interesting thing about it – point 2 is “just write what you have to write” and point 3 is a class like a normal WPF custom control. And (here is where more lines come in) – like with WPF custom controls you have to provide a style in XAML.

First of all let's talk about the data.

Instead of providing some fake data I use a complex class from my RFID project.

And for the ones interested in the data details I provided a little video about it.

In this video I also explain the styles I use to display / edit this data.

[Here you can watch the video](#) (about 40 Minutes).

For those of you which are too lazy to watch this a short description:

It's all about “AccessTime” or in other words about a class (used by the RFID Reader) which describes the times a specific RFID Transponder has access.

The Class defines days and two times (hour, minute).

For an example you can say – this Transponder has access from Monday to Friday from 9:00 to 16:45.

Further I implemented an “ObjectDataSource” – nothing more than a simple class that creates some static object of a “DataRecord” which has two members – on string (TheName) and an AccessTime as second member.

The real interesting thing is how to make the Telerik WPF RadGridView work with this data.

I also provide a video for this. But since this is the important part of this post let me first show what I did.

Or in other words – let me complete the documentation and show you the three lines of code I've been talking about.

All editors you have seen in the documentation part I pasted above have a base class called GridViewCellEditor.

We can use this class as a base class for our own editor. – This is “Line 3” of the 3 code lines and it looks like this.

```
public class GVAccessTimeEditor : GridViewCellEditor { }
```

So nothing more than a class inheriting GridViewCellEditor.

By the way – GridViewCellEditor is derived from Control – so (like with a normal custom control) we have to provide a style for it.

```
<Style TargetType="{x:Type classes:GVAccessTimeEditor}">
  <Setter Property="Template">
    <Setter.Value>
      <ControlTemplate TargetType="{x:Type classes:GVAccessTimeEditor}">
        <Grid/>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

Now we have the “Editor Control” but to assign this editor to the RadGridView we need to have an assigned “EditorSettings” class.

The reason – the Telerik RadGridView uses this EditorSettings to generate a CellEditor instance.

BUT – the good news – I'm talking about “Code Line 2” – it is nothing more than an empty class definition.

```
public class GVAccessTimeEditorSettings : EditorSettings<GVAccessTimeEditor> { }
```

As you can see this class inherits from the (templated) EditorSettings class. This assigns our editor class to this EditorSettings.

Now we can assign this EditorSettings (and with this the Editor itself) to those columns in the RadGridView which are bound to a member of the type AccessTime. – In this Example the RadGridView has the name rgvSample.

```
foreach(GridViewDataColumn gvd in rgvSample.Columns) {
  if(gvd.DataType == typeof(AccessTime)) {
    gvd.EditorSettings = new GVAccessTimeEditorSettings();
  }
}
```

Yes you are right – this is neither one line of code – nor has it something to do with “automatic assignment”.

I just showed this code as a possibility.

The promised “one line” looks like this.

```
rgvSample.RegisterDefaultEditor(typeof(AccessTime), typeof(GVAccessTimeEditorSettings));
```

This RadGridView method has two parameters – first the type of data the editor is built for. And the second parameter is the type of the EditorSettings which shall be used to edit this data.

After this call the rest is done automatically! 3 Lines of code – that’s all you need to get basic functionality.

Of course there is a little bit more to do if want to have the full feature set (like “undo edit” and so on). But this is common code and has more to do with “custom control development” than with “telerik needs it” code.

The first thing (like with standard controls) our template has to provide some “PARTs” – or in detail one part. This part defines the “editor zone” – or in other words that part of our control which holds the data.

Of course we can avoid this – but on the other side this would mean that we have to do the things “by hand” instead of using the internal provided mechanisms. The part name has to be: PART\_EditorControl. If the RadGridView finds such a part it assigns it to a member called EditorElement. So lets change our Style to implement this

```
<Style TargetType="{x:Type classes:GVAccessTimeEditor}">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="{x:Type classes:GVAccessTimeEditor}">
                <Grid x:Name="PART_EditorControl" DataContext="{TemplateBinding Value}" Margin="3" >
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="35"/>
                        <ColumnDefinition Width="35"/>
                    </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="50"/>
                        <RowDefinition Height="50"/>
                    </Grid.RowDefinitions>
                    <CheckBox Grid.Column="0" IsChecked="{Binding MondayAllowed}" Content="Mo" />
                    <CheckBox Grid.Column="1" IsChecked="{Binding TuesdayAllowed}" Content="Tu" />
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

You may have notice that I did also provide some binding here – Template Binding Value – this assigns the bound data member to our control. And (as with a custom control) we can now override the OnApplyTemplate method to access the data and do some initialization which we need to provide “undo functionality”. The solution I choose is simply to assign the value to an internal AccessTime member. So we can work on a copy – if the user cancels editing we have to do nothing because we never changed the original data. I called the internal AccessTime instance MyAccessTime.

```
public override void OnApplyTemplate() {
    base.OnApplyTemplate();
    MyAccessTime.StringVal = (Value as AccessTime).StringVal;
    (this.EditorElement as Grid).DataContext = MyAccessTime;
}
```

The code here first calls the OnApplyTemplate of the base class. There the assignment of our “PART\_EditorControl” to EditorElement takes place. Of course – you can also do the template handling in your code if you wish. You should do this if

- the binding is to complex to be done with an assignment to one part of your control
- if you hate to give a part of your control the name “PART\_EditorControl” 😊

What I do in my code is (beside of some casts) assign the content of the Value to my internal AccessTime instance and then bind this temporary element to my control.

Having this done - “undo edit” works always – even if you don’t want it 🐱

This means there is one thing missing – we have to assign our changed data to the Value property. Again we have a method we can overload for this – it’s named “UpdateEditorValue” and get’s called when the editing is committed.

```
protected override void UpdateEditorValue() {
    base.UpdateEditorValue();
    AccessTime atOld = Value as AccessTime;
    if(atOld.StringVal != MyAccessTime.StringVal) {
        atOld.StringVal = MyAccessTime.StringVal;
    }
}
```

Once again very simple code. First we call the base implementation then we check if our temporary value is different than the original (changes have been made) and in this case we simply assign our temporary data to the Value property.

[Here is the \(main\) video which shows the things I did here in more detail.](#) (40 Minutes)

Since I did some data binding very specific (to keep the things as simple as possible) I provide an additional video which shows how to extend data binding. [If you are curious - you can find the video here](#) (14 Minutes)

I’m sure Telerik will also cover this topic in their documentation. Till then this should give you enough information to implement your own custom editor and have the RadGridView automatically using it for your data types.