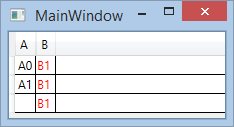
Title: <http://stackoverflow.com/questions/33200227/show-on-a-datagrid-in-red-changed-text-in-a-datatable-wpf>

How can I show in red text the value of a cell in a DataTable, when it is different from the same cell in another DataTable? In the final application the tables will be generated from changing csv files. So I have to replace autogenerated columns. I understood that you need a DataGridTemplateColumn for that, setting the CellTemplate to a resource. However this replaced column then is not part of the visual tree and therefore binding doesn't work.

http://www.thomaslevesque.com/2011/03/21/wpf-how-to-bind-to-data-when-the-datacontext-is-not-inherited/ shows that a converter implementing the Freezable object should solve that. Trying to figure out this solution, I made next simplified example. But it displays the last value of the cell in all cells of the column where the change occurs, and all in red. What am I doing wrong?



The ViewModel:

using System;

using System.ComponentModel;

using System.Data;

namespace WpfApplication1

{

class ViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged(String info)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(info));

}

}

//private Model \_Model; //for clarity left out

private DataTable \_propDataTable;

public DataTable propDataTable

{

get { return \_propDataTable; }

set

{

\_propDataTable = value;

NotifyPropertyChanged("propDataTable");

}

}

private DataTable propCopyDataTable;

private string \_sB;

public string sB

{

get { return \_sB; }

set

{

\_sB = value;

NotifyPropertyChanged("sB");

}

}

private bool \_bB = false;

public bool bB

{

get { return \_bB; }

set

{

\_bB = value;

NotifyPropertyChanged("bB");

}

}

public ViewModel()

{

propDataTable = new DataTable();

propDataTable.Columns.Add("A", typeof(string));

propDataTable.Columns.Add("B", typeof(string));

DataRow row0 = propDataTable.NewRow();

DataRow row1 = propDataTable.NewRow();

row0[0] = "A0";

row0[1] = "B0";

row1[0] = "A1";

row1[1] = "B1";

propDataTable.Rows.Add(row0);

propDataTable.Rows.Add(row1);

propCopyDataTable = propDataTable.Copy();

//now set a different value in propCopyDataTable

propCopyDataTable.Rows[1][1] = "Changed";

//find out which cells in column B are different

//try to show in red text which cell changed

for (int i = 0; i < propDataTable.Rows.Count; i++)

{

DataRow dr = propDataTable.Rows[i];

DataRow drc = propCopyDataTable.Rows[i];

sB = (string) dr["B"];

if (dr["B"].ToString().Equals(drc["B"].ToString()))

{

bB = true;

}

else

{

bB = false;

}

}

}

}

}

The ObjectToForegroundConverter:

using System;

using System.Globalization;

using System.Windows;

using System.Windows.Data;

using System.Windows.Media;

namespace WpfApplication1

{

[ValueConversion(typeof(object), typeof(SolidColorBrush))]

public class ObjectToForegroundConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

SolidColorBrush b = new SolidColorBrush(Colors.Black);

try

{

bool changedValue = (bool)value;

if (changedValue)

{

b = Brushes.Red;

}

}

catch (Exception e)

{

MessageBox.Show(string.Format("Error: {0}", e));//instance not set to a etc.

}

return b;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

}

The BindingProxy converter:

using System.Windows;

namespace WpfApplication1

{

public class BindingProxy : Freezable

{

protected override Freezable CreateInstanceCore()

{

return new BindingProxy();

}

public object Data

{

get { return (object)GetValue(DataProperty); }

set { SetValue(DataProperty, value); }

}

public static readonly DependencyProperty DataProperty =

DependencyProperty.Register("Data", typeof(object), typeof(BindingProxy), new UIPropertyMetadata(null));

}

}

The XAML control:

<Window x:Class="WpfApplication1.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:myViewModel="clr-namespace:WpfApplication1"

Title="MainWindow" Height="350" Width="525">

<Window.DataContext>

<myViewModel:ViewModel/>

</Window.DataContext>

<Window.Resources>

<myViewModel:ObjectToForegroundConverter x:Key="MyObjectToForegroundConverter"/>

<myViewModel:BindingProxy x:Key="proxy" Data="{Binding}" />

<DataTemplate x:Key="changedBColumn" >

<TextBlock

Text="{Binding Data.sB,Source={StaticResource proxy},Mode=OneWay}"

Foreground="{Binding Data.bB,Converter={StaticResource MyObjectToForegroundConverter},Source={StaticResource proxy},Mode=OneWay}"

/>

</DataTemplate>

</Window.Resources>

<Grid>

<DataGrid x:Name="myXAMLtable" AutoGeneratingColumn="DataGrid\_AutoGeneratingColumn"

ItemsSource="{Binding propDataTable}">

</DataGrid>

</Grid>

</Window>

The code behind:

using System.Windows;

using System.Windows.Controls;

namespace WpfApplication1

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

private void DataGrid\_AutoGeneratingColumn(object sender, DataGridAutoGeneratingColumnEventArgs e)

{

switch (e.Column.Header.ToString())

{

case "B":

{

DataGridTemplateColumn BTemplateColumn = new DataGridTemplateColumn();

BTemplateColumn.Header = "B";

BTemplateColumn.CellTemplate = (DataTemplate)Resources["changedBColumn"];

e.Column = BTemplateColumn;

break;

}

}

}

}

}

|  |  |
| --- | --- |
| up vote0down vote[accept](javascript:void(0);) | The problem here is each cell need a bool to indicate the state of changing. But you have just 1 property for a column in your view-model. That means all the cells in the same column will have the same state depending on that property. That explains why all the cells in the column B are red because the property bB is set to true and bound to all cells in the column B.  You need an item class (instead of just simple string value) to hold the state, something like this:  public class Item {  public string Value {get;set;}  public Item(string value){  Value = value;  }  public bool IsChanged { get; private set;}  public void SetChanged(){  IsChanged = true;  }  public override string ToString(){  return Value;  }  public override bool Equals(object other){  var item = other as Item;  if(item == null) return false;  return item.Value == Value;  }  public override int GetHashCode(){  if(Value == null) return base.GetHashCode();  return Value.GetHashCode();  }  }  Now your DataTable should be created like this:  propDataTable = new DataTable();  propDataTable.Columns.Add("A", typeof(Item));  propDataTable.Columns.Add("B", typeof(Item));  DataRow row0 = propDataTable.NewRow();  DataRow row1 = propDataTable.NewRow();  row0[0] = new Item("A0");  row0[1] = new Item("B0");  row1[0] = new Item("A1");  row1[1] = new Item("B1");  propDataTable.Rows.Add(row0);  propDataTable.Rows.Add(row1);  propCopyDataTable = propDataTable.Copy();  //now set a different value in propCopyDataTable  propCopyDataTable.Rows[1][1] = new Item("Changed");  //find out which cells in column B are different  //try to show in red text which cell changed  for (int i = 0; i < propDataTable.Rows.Count; i++) {  DataRow dr = propDataTable.Rows[i];  DataRow drc = propCopyDataTable.Rows[i];  if (!object.Equals(dr["B"], drc["B"])) {  (dr["B"] as Item).SetChanged();  }  }  Now you also need to modify your converter like this:  public object Convert(object value, Type targetType, object parameter, CultureInfo culture)  {  SolidColorBrush b = new SolidColorBrush(Colors.Black);  var item = (Item)value;  if (item.IsChanged) {  b = Brushes.Red;  }  return b;  }  The XAML should also be modified like this:  <DataTemplate x:Key="changedBColumn" >  <TextBlock Text="{Binding [B], Mode=OneWay}"  Foreground="{Binding [B],Converter={StaticResource MyObjectToForegroundConverter},  Mode=OneWay}"  />  </DataTemplate>  NOTE: You don't need to use a proxy **unless** you want to bind the controls in DataTemplate to the main view-model. But actually that's wrong. What actually you need to bind your controls to here is each data row. In each cell template, the implicit DataContext is actually a DataRowView. So you don't need to set the Source explicitly. Just bind it normally. In the above XAML, I use the Path [B] meaning the indexer [] with a string key B is passed in directly - it is equivalent to the call someDataRowView["B"]. Also the Converter has the passed-in value as an Item (not a bool). Every time changing a value for a cell, you need to set it to a new Item, don't simply change the Item.Value property because it does not support INotifyPropertyChanged. Although in this case looks like you simply want to test it.  If you want to test the proxy technique, try creating another example. In fact that technique **does**work in your original code, when it successfully binds all cells to the bB property of your main view-model which is passed in via the proxy. |

Thanks for this comprehensible explanation. it works great!

Check for System.DBNul