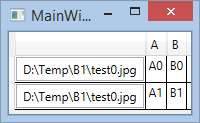
Question about images in grid.

Title: How to bind datagridcells in a template to nested objects.

The real plug-in for a MVVM application is based on a csv file in a shared DropBox. One of the columns (in below simplified example "B") might have cell values that need explanation with images. Therefore we add a folder if needed, named with the cell value ("B1" in the example), containing images. Eventually these images should become thumbnails, and full screen images if a thumbnail is selected. But for now it would be great if the added column could display the file paths belonging to a cell value in column "B" in an adjacent cell. However after days of research and trial and error the result is:



While the cell in front of A0 should be empty and in front of A1 should read:

D:\Temp\B1\test0.jpg

D:\Temp\B1\test1.jpg

I have next questions:

Why do I have to set the DataContext again for the ListBox and for the TextBlock? It was already set for the Window. Are they not in the logical tree?

Why is only the first file displayed and why in all cells of the column?

What am I doing wrong?

The View:

<Window x:Class="WpfApplication3.MainWindow"

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

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

xmlns:myViewModel="clr-namespace:WpfApplication3"

Title="MainWindow" Height="120" Width="200">

<Window.Resources>

<myViewModel:testConverter x:Key="myTestConverter"/>

</Window.Resources>

<Window.DataContext>

<myViewModel:ViewModel/>

</Window.DataContext>

<Grid>

<DataGrid x:Name="myXAMLtable" AutoGenerateColumns="True" CanUserAddRows="False"

ItemsSource="{Binding PropDataTable}">

<DataGrid.Columns>

<DataGridTemplateColumn>

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<ListBox x:Name="folder" ItemsSource="{Binding MyImageFolderList,

Converter={StaticResource myTestConverter}}">

<ListBox.DataContext>

<myViewModel:ViewModel/>

</ListBox.DataContext>

<ListBox.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding MyImageFolderList/MyImageList/MyImagePath}">

<TextBlock.DataContext>

<myViewModel:ViewModel/>

</TextBlock.DataContext>

</TextBlock>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

</DataGrid.Columns>

</DataGrid>

</Grid>

</Window>

The ViewModel:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

namespace WpfApplication3

{

class ViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

public 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 List<MyImageFolder> myImageFolderList;

public List<MyImageFolder> MyImageFolderList

{

get { return myImageFolderList; }

set

{

myImageFolderList = value;

NotifyPropertyChanged("MyImageFolderList");

}

}

public ViewModel()

{

DataTable tempPropDataTable = new DataTable();

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

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

DataRow row0 = tempPropDataTable.NewRow();

DataRow row1 = tempPropDataTable.NewRow();

row0[0] = "A0";

row0[1] = "B0";

row1[0] = "A1";

row1[1] = "B1";

tempPropDataTable.Rows.Add(row0);

tempPropDataTable.Rows.Add(row1);

PropDataTable = tempPropDataTable;

MyImageFolderList = new List<MyImageFolder>();

//in D:\Temp\B1 there are two filesP test0.jpg and test1.jpg

string B0 = "D:\\Temp\\B1\\test0.jpg";

string B1 = "D:\\Temp\\B1\\test1.jpg";

MyImageFolder mif = new MyImageFolder("B1");

MyImage mi0 = new MyImage(B0);

MyImage mi1 = new MyImage(B1);

mif.MyImageList = new List<MyImage>();//did you forget this???

mif.MyImageList.Add(mi0);

mif.MyImageList.Add(mi1);

MyImageFolderList.Add(mif);

}

}

}

The MyImageFolder class:

using System;

using System.Collections.Generic;

using System.ComponentModel;

namespace WpfApplication3

{

public class MyImageFolder : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged(String info)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(info));

}

}

private string myImageFolderPath = "";

public string MyImageFolderPath

{

get { return myImageFolderPath; }

set

{

myImageFolderPath = value;

NotifyPropertyChanged("MyImageFolderPath");

}

}

private List<MyImage> myImageList = new List<MyImage>();

public List<MyImage> MyImageList

{

get { return myImageList; }

set

{

myImageList = value;

NotifyPropertyChanged("MyImageList");

}

}

public MyImageFolder(string fp)

{

this.MyImageFolderPath = fp;

}

}

}

The MyImage class:

public class MyImage : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private void NotifyPropertyChanged(String info)

{

if (PropertyChanged != null)

{

PropertyChanged(this, new PropertyChangedEventArgs(info));

}

}

private string myImagePath = "";

public string MyImagePath

{

get { return myImagePath; }

set

{

myImagePath = value;

NotifyPropertyChanged("MyImagePath");

}

}

//constructor

public MyImage(string ip)

{

MyImagePath = ip;

}

}

The answer was:

Everywhere you are writing something like

<ListBox.DataContext>

<myViewModel:ViewModel/>

</ListBox.DataContext>

You are creating a new instance of the ViewModel class and setting it to the DataContext behind the UI object. This is almost never what you want, because the end result is something like this (note the many instances of the ViewModel object) :

<DataGridRow DataContext=ViewModel1.PropDataTable[0]>

<ListBoxItem DataContext=ViewModel2.MyImageFolderList[0]>

<TextBlock DataContext=ViewModel3.MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel2.MyImageFolderList[1]>

<TextBlock DataContext=ViewModel4.MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel2.MyImageFolderList[2]>

<TextBlock DataContext=ViewModel5.MyImageFolderPath />

</ListBoxItem>

</DataGridItem>

<DataGridRow DataContext=ViewModel1.PropDataTable[1]>

<ListBoxItem DataContext=ViewModel6.MyImageFolderList[0]>

<TextBlock DataContext=ViewModel7.MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel6.MyImageFolderList[1]>

<TextBlock DataContext=ViewModel8.MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel6.MyImageFolderList[2]>

<TextBlock DataContext=ViewModel.MyImageFolderPath />

</ListBoxItem>

</DataGridRow>

The DataContext is inherited from the parent control, so you should write you bindings with that assumption.

What you want is ONE copy of your ViewModel object, which as a property called PropDataTable that is a collection of objects. Each object in PropDataTable should have a collection of objects called MyImageFolderList, and each item in MyImageFolderList should have a property called MyImageFolderPath.

So if you have the following (note the lack of assigning .DataContext to a new instance of ViewModel)

<DataGrid ItemsSource="{Binding PropDataTable"}>

...

<ListBox ItemsSource="{Binding MyImageFolderList"}>

...

<TextBlock Text="{Binding MyImageFolderPath}" />

...

</ListBox>

</DataGrid>

It actually renders as something along these lines :

<DataGridRow DataContext=ViewModel1.PropDataTable[0]>

<ListBoxItem DataContext=ViewModel1.PropDataTable[0].MyImageFolderList[0]>

<TextBlock DataContext=ViewModel1.PropDataTable[0].MyImageFolderList[0].MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel1.PropDataTable[0].MyImageFolderList[1]>

<TextBlock DataContext=ViewModel1.PropDataTable[0].MyImageFolderList[1].MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel1.PropDataTable[0].MyImageFolderList[2]>

<TextBlock DataContext=ViewModel1.PropDataTable[0].MyImageFolderList[2].MyImageFolderPath />

</ListBoxItem>

</DataGridItem>

<DataGridRow DataContext=ViewModel1.PropDataTable[1]>

<ListBoxItem DataContext=ViewModel1.PropDataTable[1].MyImageFolderList[0]>

<TextBlock DataContext=ViewModel1.PropDataTable[1].MyImageFolderList[0].MyImageFolderPath />

</ListBoxItem>

<ListBoxItem DataContext=ViewModel1.PropDataTable[1].MyImageFolderList[1]>

<TextBlock DataContext=ViewModel1.PropDataTable[1].MyImageFolderList[1].MyImageFolderPath />

</ListBoxItem>

</DataGridRow>

I have an article on my blog that goes into the DataContext in a bit more detail if you're struggling to understand it as well. [What is this "DataContext" you speak of?](https://rachel53461.wordpress.com/2012/07/14/what-is-this-datacontext-you-speak-of/)

|  |  |
| --- | --- |
| [share](http://stackoverflow.com/a/33416651/2406292)[edit](http://stackoverflow.com/posts/33416651/edit) | answered 19 mins ago  [[https://www.gravatar.com/avatar/01b235a8024b2dd10f427c301c876a0b?s=32&d=identicon&r=PG](http://stackoverflow.com/users/302677/rachel)Rachel](http://stackoverflow.com/users/302677/rachel) **63.8k**28167280 |

Thanks a lot. Nice blog also! Do I understand it right if I say that by inserting a tag inside a tag you go down the Data layer tree when binding? The PropDataTable was just filled with strings. But when I change them into objects after a day of trial and error I still have empty cells. When I try to set the binding through the Properties window of VS it also doesn't show the properties I need. Strangely it shows a tree: ViewModel – PropDataTable – PropDataTable – DataView etc.

Is the following reasoning ok?

So TextBox should be bound to string property: string \*\*ImagePath\*\* of object\*\*Image\*\*. The parent of TextBox (ListBox) is bound to property: List<Image>\*\*ImageList\*\*. The parent of ListBox (DataGrid) is bound to the property: DataTable \*\*PropDataTable\*\* which contains the objects that has the ImageList property. I call these objects \*\*Item\*\*. So \_Item\_ has a property List<Image> \*\*ImageList\*\* and \_Image\_ has a property string \*\*ImagePath\*\*. If the reasoning is right, what can be wrong with my code? I have put it on GitHub [link]( https://github.com/jhubers/test)

Answer of Rachel:

@H.H. PropDataTable is a DataTable object, and DataTable does not have a property called ImageList. Change PropDataTable to a collection of objects which contain a public List<Image> property calledImageList – [Rachel](http://stackoverflow.com/users/302677/rachel) [2 hours ago](http://stackoverflow.com/questions/33415736/how-to-bind-datagridcells-in-a-template-to-nested-objects#comment54692182_33416651).

My comment:

I don't get this. If PropDataTable is not a DataTable, how can I use it to make a DataGrid? Or is every row of the DataGrid filled with the objects in the collection until the row is full and then the next row? Also in the real plug-in I don't know how many columns there are in the changing csv file...

I could of course automatically generate the columns on the basis of PropDataTable and replace them in the code behind the xaml. And then use PropDataTable to make a collection. It would be good if the imagepaths are also written to the History file. But a list in an Excel cell? You use a softreturn for that. What does it looks like in csv? Test shows that then the content of the cell is put between "" and normal returns between the lines. How can I handle this in C#? If I try it in pCOLAD I get an error about primary keys in the datatable (Figure 1). Of course because there are no unique values in column Parameter in row 7 and 8. Row 7,8 and 9 should be one row. Is that possible?

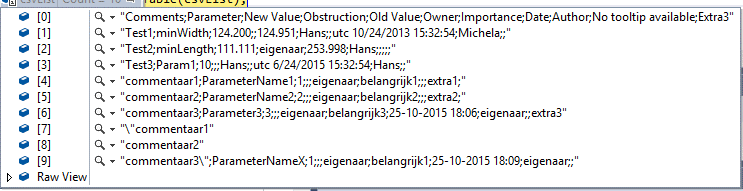


Figure 1 csv file with softreturns in row 7 loaded in DataTable

The problen lies in: line = myStream.ReadLine(); you\ll have to test if there are returns between the "". On the [msdn site](https://msdn.microsoft.com/en-us/library/vstudio/system.io.streamreader.readline(v=vs.100).aspx) an alternative is presented:

function code:

string ReadNextMultiline(StreamReader mlReader)  
        {  
            bool MultilineDetected;  
            string res = "", mLine = "";              
            do  
            {  
                MultilineDetected = false;  
                mLine = mlReader.ReadLine();  
                res = String.Concat(  
                                        res,   
                                        (res.Length > 0?" ":""),    // add a space where there was a linebreak.  
                                        mLine);  
                string[] broken = res.Split(';');  
                // if the RES features unfinished multiliner, then the LAST element will contain exactly 1 " symbol:  
                if ((broken[broken.Length - 1].IndexOf('\"') >= 0) &&               // there's some " symbol  
                    (broken[broken.Length - 1].IndexOf('\"') == broken[broken.Length - 1].LastIndexOf('\"'))    // there's exactly 1 " on that row.  
                   )  
                {  
                    MultilineDetected = true;  
                }  
            } while (MultilineDetected);  
            return res;                          
        }

That works, but have to apply the recommended style for streamreader:

using (StreamReader sr = new StreamReader(filename))

{

string line = "";

while (sr.Peek() >= 0)

{

line = ReadNextMultiline(sr);

csvList.Add(line);

}

sr.Dispose();

return csvList;

}

And you have to replace the adde space by "\n".

Still the problems stay with displaying the result. Next comments on stackoverflow site lead to the answer.

Usually I would recommend making the property a List<T> of MyClass objects, where MyClass contains public properties of the fields you want, however since the columns aren't known at runtime and could change, a DataTable might be a bit better. Try manually adding an extra column to your DataTable with the fieldname of ImageList... if you can't set the datatype of the column to a List<Image> then you might need to leave it as an object and use an IValueConverter to convert it from object to List<Image> in the binding – [Rachel](http://stackoverflow.com/users/302677/rachel) [Oct 31 at 21:43](http://stackoverflow.com/questions/33415736/how-to-bind-datagridcells-in-a-template-to-nested-objects#comment54700309_33416651)

I can add a column to the DataTable of type List<Image>, but the datagrid then shows (Collection) in the cell. How do I get to the content of the collection? – [H.H.](http://stackoverflow.com/users/2406292/h-h) [yesterday](http://stackoverflow.com/questions/33415736/how-to-bind-datagridcells-in-a-template-to-nested-objects#comment54776814_33416651)

You'll have to use a DataGridTemplateColumn like you have in your original question, and I would probably use an IValueConverter to convert the DataRow (which will be the DataContext of your Template) to your List<Image>... it sounds like your grid will have variable columns, but the List<Image> will always be static. Btw if your grid does NOT have variable columns, I would highly recommend making a custom class for it instead of using the DataTable – [Rachel](http://stackoverflow.com/users/302677/rachel) [yesterday](http://stackoverflow.com/questions/33415736/how-to-bind-datagridcells-in-a-template-to-nested-objects#comment54788578_33416651)

var item = (DataRowView)value; return item[0]; this works. Thanks. – [H.H.](http://stackoverflow.com/users/2406292/h-h) [yesterday](http://stackoverflow.com/questions/33415736/how-to-bind-datagridcells-in-a-template-to-nested-objects#comment54793793_33416651)

The final version can be found here: [Link](https://github.com/jhubers/test) – [H.H.](http://stackoverflow.com/users/2406292/h-h) [21 hours ago](http://stackoverflow.com/questions/33415736/how-to-bind-datagridcells-in-a-template-to-nested-objects#comment54821399_33416651).

Later I added also the showing of a full screen image that is zoomable like in Revit and Dynamo:

The full screen control:

<Window x:Class="WpfApplication3.FullScreenImage"

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

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

xmlns:local="clr-namespace:WpfApplication3"

SizeToContent="WidthAndHeight"

Title="FullScreenImage" >

<Grid>

<local:ZoomBorder x:Name="border" ClipToBounds="True" Background="Gray">

<Image Name="fullImage" >

</Image>

</local:ZoomBorder>

</Grid>

</Window>

The code behind:

using System.Windows;

namespace WpfApplication3

{

public partial class FullScreenImage : Window

{

public FullScreenImage()

{

InitializeComponent();

}

}

}

The ZoomBorder:

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

namespace WpfApplication3

{

public class ZoomBorder : Border

{

private UIElement child = null;

private Point origin;

private Point start;

private TranslateTransform GetTranslateTransform(UIElement element)

{

return (TranslateTransform)((TransformGroup)element.RenderTransform)

.Children.First(tr => tr is TranslateTransform);

}

private ScaleTransform GetScaleTransform(UIElement element)

{

return (ScaleTransform)((TransformGroup)element.RenderTransform)

.Children.First(tr => tr is ScaleTransform);

}

public override UIElement Child

{

get { return base.Child; }

set

{

if (value != null && value != this.Child)

this.Initialize(value);

base.Child = value;

}

}

public void Initialize(UIElement element)

{

this.child = element;

if (child != null)

{

TransformGroup group = new TransformGroup();

ScaleTransform st = new ScaleTransform();

group.Children.Add(st);

TranslateTransform tt = new TranslateTransform();

group.Children.Add(tt);

child.RenderTransform = group;

child.RenderTransformOrigin = new Point(0.0, 0.0);

this.MouseWheel += child\_MouseWheel;

this.MouseDown += child\_MouseDown;

this.MouseUp += child\_MouseUp;

this.MouseMove += child\_MouseMove;

this.PreviewMouseRightButtonDown += new MouseButtonEventHandler(

child\_PreviewMouseRightButtonDown);

}

}

public void Reset()

{

if (child != null)

{

// reset zoom

var st = GetScaleTransform(child);

st.ScaleX = 1.0;

st.ScaleY = 1.0;

// reset pan

var tt = GetTranslateTransform(child);

tt.X = 0.0;

tt.Y = 0.0;

}

}

#region Child Events

private void child\_MouseWheel(object sender, MouseWheelEventArgs e)

{

if (child != null)

{

var st = GetScaleTransform(child);

var tt = GetTranslateTransform(child);

double zoom = e.Delta > 0 ? .2 : -.2;

if (!(e.Delta > 0) && (st.ScaleX < .4 || st.ScaleY < .4))

return;

Point relative = e.GetPosition(child);

double abosuluteX;

double abosuluteY;

abosuluteX = relative.X \* st.ScaleX + tt.X;

abosuluteY = relative.Y \* st.ScaleY + tt.Y;

st.ScaleX += zoom;

st.ScaleY += zoom;

tt.X = abosuluteX - relative.X \* st.ScaleX;

tt.Y = abosuluteY - relative.Y \* st.ScaleY;

}

}

private void child\_MouseDown(object sender, MouseButtonEventArgs e)

{

if (e.ChangedButton == MouseButton.Middle | e.ChangedButton == MouseButton.Left)

{

if (child != null)

{

var tt = GetTranslateTransform(child);

start = e.GetPosition(this);

origin = new Point(tt.X, tt.Y);

this.Cursor = Cursors.Hand;

child.CaptureMouse();

}

}

}

private void child\_MouseUp(object sender, MouseButtonEventArgs e)

{

if (e.ChangedButton == MouseButton.Middle | e.ChangedButton == MouseButton.Left)

{

if (child != null)

{

child.ReleaseMouseCapture();

this.Cursor = Cursors.Arrow;

}

}

}

void child\_PreviewMouseRightButtonDown(object sender, MouseButtonEventArgs e)

{

this.Reset();

}

private void child\_MouseMove(object sender, MouseEventArgs e)

{

if (child != null)

{

if (child.IsMouseCaptured)

{

var tt = GetTranslateTransform(child);

Vector v = start - e.GetPosition(this);

tt.X = origin.X - v.X;

tt.Y = origin.Y - v.Y;

}

}

}

#endregion

}

}

The MainWindow xaml:

<Window x:Class="WpfApplication3.MainWindow"

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

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

xmlns:myViewModel="clr-namespace:WpfApplication3"

SizeToContent="WidthAndHeight"

Title="MyWindow">

<Window.DataContext>

<myViewModel:ViewModel/>

</Window.DataContext>

<Window.Resources>

<myViewModel:testConverter x:Key="myTestConverter"/>

<myViewModel:DataRowToListConverter x:Key="myDataRowToListConverter"/>

<DataTemplate x:Key="convertedImage">

<ListBox ItemsSource="{Binding Converter={StaticResource myDataRowToListConverter}}">

<ListBox.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding ImagePath}"/>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

</DataTemplate>

</Window.Resources>

<Grid >

<DataGrid x:Name="myXAMLtable" AutoGenerateColumns="True" CanUserAddRows="False"

ItemsSource="{Binding PropDataTable}" AutoGeneratingColumn="myXAMLtable\_AutoGeneratingColumn">

<DataGrid.Columns>

<DataGridTemplateColumn>

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<ListBox ItemsSource="{Binding Converter={StaticResource myDataRowToListConverter}}">

<ListBox.ItemTemplate>

<DataTemplate>

<Image Source="{Binding ImagePath}" Width="60"

HorizontalAlignment="Left" MouseUp="Image\_MouseUp">

</Image>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

</DataGrid.Columns>

</DataGrid>

</Grid>

</Window>

The code behind MainWindow:

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

namespace WpfApplication3

{

/// <summary>

/// Interaction logic for MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

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

{

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

{

case "Images":

// Create a new template column.

DataGridTemplateColumn imageTemplateColumn = new DataGridTemplateColumn();

imageTemplateColumn.Header = "Images";

imageTemplateColumn.CellTemplate = (DataTemplate)Resources["convertedImage"];

// Replace the auto-generated column with the templateColumn.

e.Column = imageTemplateColumn;

e.Column.Width = 200;

break;

default:

e.Column.Width = 100;

break;

}

}

private void Image\_MouseUp(object sender, MouseButtonEventArgs e)

{

var im = (System.Windows.Controls.Image)sender;

FullScreenImage myFullScreenImage = new FullScreenImage();

myFullScreenImage.fullImage.Source = im.Source;

myFullScreenImage.Show();

}

}

}

The DataRowToListConverter:

using System;

using System.Collections.Generic;

using System.Data;

using System.Globalization;

using System.Windows.Data;

namespace WpfApplication3

{

[ValueConversion(typeof(DataRowView), typeof(List<Image>))]

class DataRowToListConverter : IValueConverter

{

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

{

List<Image> li = new List<Image>();

if (value==null)

{

return li;

}

else

{

var item = (DataRowView)value;

return item[0];

}

}

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

{

throw new NotImplementedException();

}

}

}

The ViewModel:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

namespace WpfApplication3

{

class ViewModel : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

public 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");

}

}

//constructor

public ViewModel()

{

PropDataTable = new DataTable();

Image B1I0 = new Image("D:\\Temp\\B1\\test0.jpg");

Image B1I1 = new Image("D:\\Temp\\B1\\test1.jpg");

List<Image> B1L = new List<Image>();

B1L.Add(B1I0);

B1L.Add(B1I1);

Item A0 = new Item("A0");

Item B0 = new Item("B0");

Item A1 = new Item("A1");

Item B1 = new Item("B1");

PropDataTable.Columns.Add("Images", typeof(List<Image>));

PropDataTable.Columns.Add("A", typeof(Item));

PropDataTable.Columns.Add("B", typeof(Item));

DataRow row0 = PropDataTable.NewRow();

DataRow row1 = PropDataTable.NewRow();

row0[1] = A0;

row0[2] = B0;

row1[1] = A1;

row1[2] = B1;

row1[0] = B1L;

PropDataTable.Rows.Add(row0);

PropDataTable.Rows.Add(row1);

}

}

}

The Item:

namespace WpfApplication3

{

public class Item

{

public string Svalue { get; set; }

//overiding the ToString method takes care of displaying the string value in the DataGrid

public override string ToString()

{

return Svalue;

}

//constructor

public Item(string s)

{

Svalue = s;

}

}

}

The Image:

namespace WpfApplication3

{

public class Image

{

public string ImagePath { get; set; }

public Image(string ip)

{

ImagePath = ip;

}

}

}