Rotating picture display

Your pictures that reside on your computer disk and network disks

**Source code location:** <https://github.com/avifarah/Rotate.Pictures>

# Introduction

The purpose of this solution is to retrieve files in directories and subdirectories that you provide and display them, continuously, in a random order.

## Description of the interface



Figure 1

The display is composed of

* Menu, see top left
* Picture, the majority of the display
* Bottom bar appearing mostly in yellow background, contains both information and allows actions on the picture rotation set itself

The Menu items allow you to change all the factors provided in the configuration file.

The picture is “clickable”, mouse selection of the picture toggles the rotate / hold-still mode of picture rotation. The rotation mode is accompanied with an indicator represented as a traffic-light circle kind of indicator, in the bottom bar. Green for rotating automatically mode, Red for holding-still. The green/red traffic-light indicator is “clickable” as well and has the same effect as clicking on the picture itself.

 (*Figure 2*) for rotating mode, and  (*Figure 3*) for still mode.

The bottom bar allows you to move back and forth through the pictures. Forward in a random picture selection and backwards in the order they were displayed, using the left arrow and the right arrow at each end of the bottom bar. You saw these arrows in Figure 1 and the right arrow in the above two pictures depicting the traffic light indicator for state. The Left arrow is on the other side of the picture, at the bottom, .(*Figure 4*) The bottom bar also contains the full path of the picture displayed. This path can be selected, using the mouse and copied with the keyboard combination of <Ctrl>+<c>.

The red-circle-with-a-backslash-line-through-it-on-a-white-background button, the third icon from the right, in *Figure 2* and in *Figure 3*, allows you to select a picture that you wish not to see again.

# Preparing for Installation

Modify the App.config file to make the changes permanent. Modify the App.config by modifying the following, value, entries in the appSettings section:

<appSettings> <!-- Directories where the system will look for pictures to display.  
  Directories are semicolon separated.  Ex: value="c:\pic;g:\pic;m:\pic" -->  
 <add key="Initial Folders" value="C:\Pictures"/>  
  
 <!-- Depth of stack keeping the displayed pictures, meaning you may go   
  back up to "Max picture tracker depth" of displayed pictures -->  
 <add key="Max picture tracker depth" value="1000"/>  
  
 <!-- These are the only extensions that the system will consider.  
  Extensions are semicolon separated.  
 Extensions must start with a period (".").  -->  
 <add key="Still pictures" value=".jpg;.bmp;.gif;.png;.psd;.tif"/>  
 <add key="Motion pictures" value=".mov;.avi;.mpg;.mp4;.wmv;.3gp"/>  
  
 <!-- Image stretch may be:  
 "Fill" - Stretch the picture height and width independently  
  "None" - Original size of height and width is maintained  
 "Uniform" Stretches the height and width uniformly until the one of the   
 directions equals the height or the width of the window  
  "UniformToFill" Stretches the height and width uniformly passed the  
 the first dimension to reach the window height or width

and until the second dimension reaches the height  width of the window -->  
 <add key="Image stretch" value="Uniform"/>  
  
 <!-- Time to wait between display of one picture to the next  
  Value may contain fraction of a second.  -->  
 <add key="Timespan between pictures [Seconds]" value="15"/>  
  
 <!-- The first picture to be displayed is treated differently then the rest   
 of the pictures.  However, if you leave the value of "First picture to   
 display" blank then the system will treat the first picture like it  
  treats the rest of the picture, choose it randomly.  Though not having  
  first picture may mean that your wait for "Timespan between pictures  
  [Seconds]" before the first picture appears. -->  
 <add key="First picture to display" value="C:\Pictures\Ben\IMG\_0840-1.JPG"/>  
  
 <!-- The system will start with automatic rotation of picture if the value   
  to "On start image rotating" is true otherwise the first picture will   
  be frozen until you change the running status by selecting (click) on   
  the picture or select the forward arrow -->  
 <add key="On start image rotating" value="True"/>  
  
 <add key="Visual heartbeat" value="400" />  
 <add key="Fast Forward [Seconds]" value="10" />  
 <add key="Pictures Indices To Avoid. Comma separated" value=""/>  
 <add key="FilePath to save Pictures to avoid" value="%temp%\Rotate.Picture.lst"/>  
</appSettings>

*Listing 1*

Also modify the value entry of file type="log4net.Util.PatternString", part of the appender entry in the log4net config section (see *Listing 2* below):

<log4net>  
    <appender name="Console" type="log4net.Appender.ConsoleAppender">  
        <layout type="log4net.Layout.PatternLayout">  
            <conversionPattern value="%5level - %message%newline" />  
        </layout>  
    </appender>  
    <appender name="AppRollingFile" type="log4net.Appender.RollingFileAppender">  
        <!-- Set the value to the directory where you care for messages to go into.  
            The %date{yyyyMMdd} translates to the date value, 4 digit year,  
            2 digit month and 2 digit day.  Date value when the program started  
            executing -->  
        <file type="log4net.Util.PatternString" value="C:\Users\<your  
name>\AppData\Local\Temp\Logs\RotatePictures\RotatePictures.%date{yyyyMMdd}.log" />  
        <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />

        <appendToFile value="true" />  
        <rollingStyle value="Composite" />  
        <maximumFileSize value="3MB" />  
        <maxSizeRollBackups value="15" />  
        <layout type="log4net.Layout.PatternLayout">  
            <conversionPattern value="%-5level [%thread] (%file:%line) [%d{yyyy-MM-dd HH:mm:ss.fff}] %message%newline" />  
        </layout>  
    </appender>  
    <root>  
        <level value="All" />  
        <appender-ref ref="Console" />  
        <appender-ref ref="AppRollingFile" />  
    </root>  
</log4net>

*Listing 2*

All the changes in the **appSettings** section in the configuration file, **App.Config**, or **Rotate.Pictures.exe.Config** file (see *Listing 1* above), can be done from within the running program. The changes in section **log4net** (see *Listing 2* above) cannot be done except in the configuration file.

# Installation

Installation itself is copy-paste installation. After compiling in “Release” mode copy all of the contents of the **bin\Release** folder to where ever you wish to “install” the program. Then for your convenience you may set a screen shortcut or a taskbar shortcut.

# Running the program

From within the running the program you may change most of the meta data values through two menu item selection: ***Tools*** and ***Stretch Mode…***, see Figure 5 below:



Figure 5

## The ***Tools*** menu

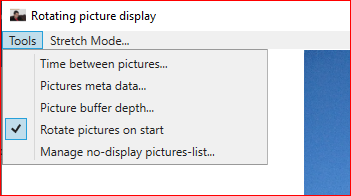


Figure 6

The ***Tools*** set of menus options, control most of the options in the configuration. We will delve into it after we look into the **Stretch Mode…** menu item.

## **Stretch Mode…** menu item

Selecting the ***Stretch Mode…*** menu item brings up a dialog containing four choices, see Figure 7 below, where the dialog is located close to the top left corner of the screen and it is showing the **Uniform** stretch mode as the selection. The picture displayed is in the **Uniform** stretch mode. The four choices in the dialog are:

* Fill
* None
* Uniform
* Uniform ToFill

**Comment:** The dialog does not have taskbar but you may drag it by selecting, holding and dragging any background area.

Setting this option is equivalent to the configuration setting of:

<add key="Image stretch" value="Fill"/>  
 <add key="Image stretch" value="None"/>  
 <add key="Image stretch" value="Uniform"/>  
 <add key="Image stretch" value="Uniform ToFill"/>

In the configuration file (see *Listing 1* above). The following picture, Figure 7, depicts the dialog box and the picture displayed in the ***Uniform*** stretch mode.

Uniform stretch mode means that the picture will grow/shrink in the same proportions in both horizontal and vertical directions as the window expands and shrinks.

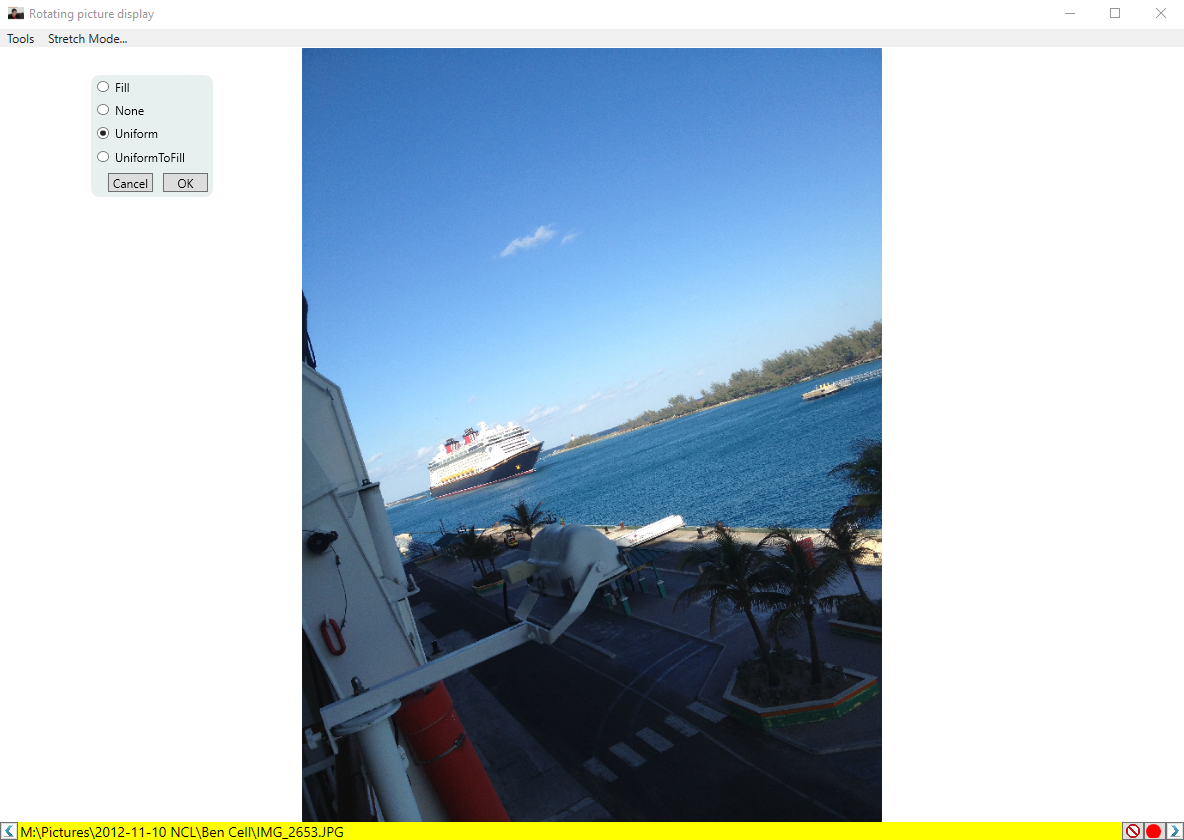


Figure 7

Setting the stretching mode to ***Fill***, will stretch the picture differently in the horizontal and vertical directions so the picture fills the entire window, even as you stretch and contract the window. See Figure 8 below:



Figure 8

Setting stretch mode to ***None*** will not stretch the picture even as you stretch the window, see Figure 9:



Figure 9

**Comment:** In this depiction you may see no difference between ***None*** and **Uniform** stretching mode. However, changing the window size will reveal difference in the behavior of the **None** and **Uniform** stretching modes.

Lastly, setting the stretching mode to ***Uniform ToFill*** will stretch the short side to fill the screen, then the original long side of the picture will be stretched, proportionally, beyond the screen boundaries. The above picture’s short side is in the horizontal direction and so the horizontal side will stretch to fill then the horizontal length of the window and the vertical side of the picture will extend proportionally beyond the display area of the window. The above picture when stretched in ***Uniform ToFill*** mode will display the following (see Figure 10):



Figure 10

## The **Tools** menu dropdown (see above Figure 6)

### ***Time between pictures…***

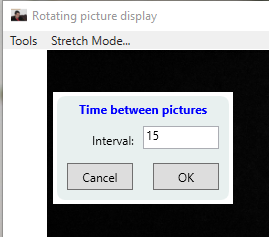


Figure 11

**Comment:** The dialog does not have taskbar but you may drag it by selecting, holding and dragging any background area.

This dialog will allow you to control the time, in seconds, between pictures. The dialog will accept neither negative numbers nor the value of 0.

This option is equivalent to configuration option:

<add key="Timespan between pictures [Seconds]" value="15"/>

(see *Listing 1* above). Setting the value in the configuration file to a non-number, 0 or a negative number, will not take effect and the program will reset the value to 10 seconds between pictures.

## ***Pictures meta data…*** (See Figure 6 of the ***Tools*** menu option dropdown)

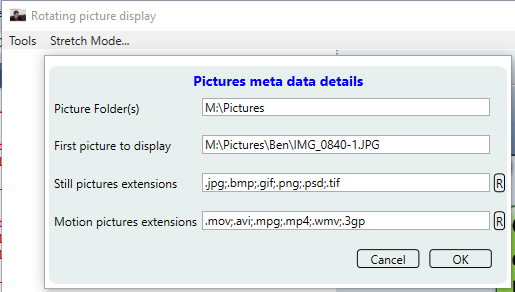


Figure 12

**Comment:** Since this menu option allows for changes in the loaded picture folder(s) this option will not be available while the program loads the picture collection information. Therefore, in the beginning and depending on the number of pictures that you have for the program to know about, the ***Picture meta data…*** option will not b/5 available.

**Comment:** The dialog does not have taskbar but you may drag it by selecting, holding and dragging any background area.

**Picture Foler(s):** is equivalent to resetting the value of configuration key “Initial Folders”:

<add key="Initial Folders" value="M:\Pictures" />

See *Listing 1* above. At this point the program will freeze the rotation of the pictures for a while and reload the listing of the pictures.

**First picture to display:** is equivalent to resetting the value of configuration key “First picture to display”:

<add key="First picture to display" value="M:\Pictures\Ben\IMG\_0840-1.JPG" />

**Still pictures extensions:** is equivalent to resetting the value of configuration key “Still pictures”

<add key="Still pictures" value=".jpg;.bmp;.gif;.png;.psd;.tif" />

At the end of the line you see a button labeled “R”. “R” will restore the value to their original value: ".jpg;.bmp;.gif;.png;.psd;.tif"

Motion pictures extensions: is equivalent to resetting the value of configuration key "Motion pictures":

<add key="Motion pictures" value=".mov;.avi;.mpg;.mp4;.wmv;.3gp" />

The “R” button at the end of the line will restore original value: ".mov;.avi;.mpg;.mp4;.wmv;.3gp"

**Comment:** When running the **OK** button of the **Pictures meta data details** dialog you will:

* Reload all pictures in the **Picture(s) folder** textbox
* Set historic picture buffer set to nil (buffer of historic pictures selection will restart)
* Set the do-not-display picture set to nil. If you wish to preserve these pictures as they are then you should first select the **Manage no-display pictures-list…** menu item and save the list to a file (see Figure 14). Then come back to the **Pictures meta data…** menu selection (that will load this dialog), then choose the **OK** button. Finally, when the pictures are loaded, select the **Manage no-display pictures-list…** and retrieve the list.

**Comment:** You should run the OK button when one of those meta data changes. Otherwise, select the cancel button. If the meta data did not change, but you added pictures or deleted pictures, you may still run through this dialog to capture the changes.

## Picture buffer depth...

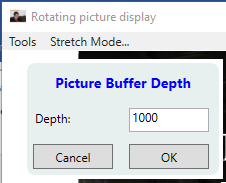


Figure 13

**Comment:** The dialog does not have taskbar but you may drag it by selecting, holding and dragging any background area.

Is equivalent to resetting the value of configuration key “Max picture tracker depth”

<add key="Max picture tracker depth" value="1000" />

Dialog will not accept a negative, 0 or non-number values.

## Rotate pictures on start

Is equivalent to resetting the value of configuration key “On start image rotating”

<add key="On start image rotating" value="True" />

## Manage no-display pictures-list...

The red-circle-with-a-backslash-line-through-it-on-a-white-background button, the third icon from the right, in *Figure 2* and in *Figure 3*, allows you to select a picture that you wish not to see again.

The **Manage no-display pictures-list…** manages that list.

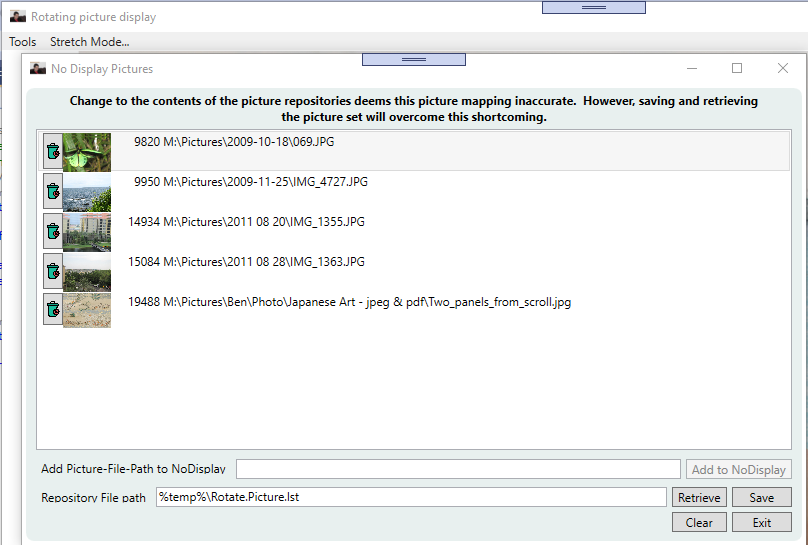


Figure 14

The menu option for displaying this dialog will not be available while the program reads the collection of pictures. This is because the index to the pictures is not complete until the collection is completely read.

The columns of the list of pictures is:

* Garbage can icon that allows the user to remove the picture from the do-not-display-list
* The picture itself
* Index of picture
* Path of picture

If the collection changes then when the user restarts the program, then the index assigned to the pictures will change. As such, the user could save the collection of paths and retrieve them. The process of retrieving the pictures will look for the appropriate indices.

Selecting the garbage can to the left of the picture will remove the picture from the do-not-display list. The program will display the path in the **Add Picture-File-Path to NoDisplay** textbox and allow the user to add that picture back to the do-not-display list (using the **Add to NoDisplay** button). Obviously, the user may add picture to the collection of do-not-display list by adding the path to the **Add Picture-File-Path to NoDisplay** textbox and selecting the **Add to NoDisplay** button.

Setting the **Repository File path** and selecting the **Save** button will save the collection of items not to be displayed any longer. The Retrieve button will restore those do-not-display items to the do-not-display collection.

The **Clear** button will clear the do-not-display selection.

# The code

The solution is a relatively simple MVVM project, resulting in an executable that is less than a 1 MB.

## Model

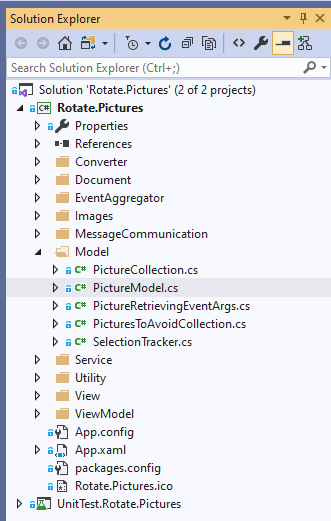


Figure 15

The model, represented by the PictureModel class, is a file name repository. PictureModel relies on the PictureCollection class that stores all the file names in the directories provided in the "Initial Folders" of the configuration file, filtered by the allowable extensions.

The model also relies on the SelectionTracker class that stores the file names that were displayed and allow for back and forth motion of pictures using the arrows in the bottom bar.

Lastly, the model also relies on the PicturesToAvoidCollection class to keep track of the pictures to avoid displaying.

The ViewModel, MainWindowViewModel, holds a reference to the model, though it does not instantiate the class. which is left to a factory:

\_model = (PictureModel)ModelFactory.Inst.Create("PictureFileRepository");

The model, PictureModel, loads the pictures asynchronously:

\_taskModel = Task.Run(() => RetrievePictures(), \_cts.Token);

Thereafter the pictures are selected in a random fashion:

var cnt = \_picCollection.Count;

var index = \_rand.Next(cnt);

return \_picCollection[index];

## ViewModel

The ViewModel, for example MainWindowViewModel drives the View, MainWindow. The ViewModel section includes other window controls, that are used to display and alter information provided in the .config file.

### Communication between the ViewModels

The purpose of the MVVM pattern is to separate concerns between the disparate logical components, so the View is responsible for the display of information only which is separate from the model that is responsible for retrieving and saving data which is separate from the “glue” between them called the ViewModel.

However, we need to communicate between the various ViewModel components and we should not let one ViewModel component hold an instance of another ViewModel component. The communication between the ViewModel components is done through a Messenger, creatively named Messenger.

The recipient of a message needs to register and potentially unregister the message. Between the time that the message is registered and before it is unregistered, any class can send the registered message to be handled by the class that registered to handle the message.

The ViewModel FileTypesToRotateViewModel registers the message SelectedMetadataMessage and processes requests as follows:

**Step 1**

Register the message:

Messenger.DefaultMessenger.Register<SelectedMetadataMessage>(

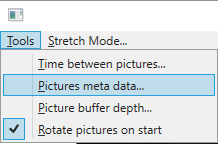
this, OnMetaDataProcess, MessageContext.SelectedMetadataViewModel);

The third (3rd) parameter, MessageContext.SelectedMetadataViewModel, is a string used as context for the message. Context, in this scenario, is a differentiating mechanism for messages.

This registering class, FileTypesToRotateViewModel, is not instantiated until the Main Window, MainWindowViewModel, asks for it be instantiated, and as such it cannot receive nor handle messages until it is instantiated.

**Step 2**

Upon selecting the menu item Tools / Pictures meta data…



The command processing for this menu item, part of the MainWindowViewModel, calls:

\_pictureMetadataService.ShowDetailDialog(metaData);

Where the parameter, metaData, is the data needed to populate the View, FileTypesToRotateView and \_pictureMetadataService is an instance of the service, FileTypeToRotateService, responsible for displaying the FileTypesToRotateView window, as a dialog. The metaData parameter will be passed on in the SelectedMetadataMessage message, after the FileTypesToRotateView window was created by the service, FileTypeToRotateService. The metaData will be sent via the following message:

Messenger.DefaultMessenger.Send(

new SelectedMetadataMessage(metadata),

MessageContext.SelectedMetadataViewModel);

**Step 3**

FileTypesToRotateViewModel’s callback method, OnMetaDataProcess (see **Step 1** registration), processes the message, in our case the callback displays the information within the window, FileTypesToRotateView.

**On the way back**

When the user selects the OK button, in the FileTypesToRotateView class, the above three (3) steps are used again to update the result of the window. The main window’s ViewModel, MainWindowViewModel, register this same SelectedMetadataMessage but a different context.

**Step 1**

Messenger.DefaultMessenger.Register<SelectedMetadataMessage>(

this, OnSetMetadataAction, MessageContext.SetMetadata);

**Step 2**

The handling of the OK button in the FileTypesToRotateViewModel sends:

Messenger.DefaultMessenger.Send(

new SelectedMetadataMessage(metadata),

MessageContext.SetMetadata);

Then still part of handling the OK button, FileTypesToRotateViewModel unregisters from the SelectedMetadataMessage message:

Messenger.DefaultMessenger.Unregister(

this,

MessageContext.SelectedMetadataViewModel);

The Unregister is necessary, otherwise every time the window FileTypesToRotateView is instantiated a new call to the register will happen which will duplicate and triplicate the handling.

**Step 3**

MainWindowViewModel handles the updates from the FileTypesToRotateViewModel.

## View

### Satisfying the MVVM equation

The views are Xaml defined controls. Each View control instantiates the appropriate ViewModel class and assigns its DataContext to it. However, the View does not instantiate the ViewModel directly, it calls a factory class.

DataContext = VmFactory.Inst.Create(this);

The VmFactory class is a simple class using reflection to instantiate the ViewModel class.

### Handling Errors

The FileTypesToRotateView needs to handle errors. The XAML file adds the following template

<Grid.Resources>

<Style TargetType="{x:Type TextBox}">

<Setter Property="Validation.ErrorTemplate">

<Setter.Value>

<ControlTemplate>

<StackPanel>

<AdornedElementPlaceholder>

<Border BorderBrush="Red" BorderThickness="2"/>

</AdornedElementPlaceholder>

<ItemsControl ItemsSource="{Binding}">

<ItemsControl.ItemTemplate>

<DataTemplate>

<TextBlock x:Name="ErrorText"

Text="{Binding ErrorContent}"

Foreground="Red"/>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</StackPanel>

</ControlTemplate>

</Setter.Value>

</Setter>

</Style>

</Grid.Resources>

Which paints the border of the TextBox containing the error red and displays the error message under the TextBox.

The TextBox that is needs to opt into the error program needs to mark its binding with ValidatesOnDataErrors=True, and potentially ValidatesOnExceptions=True, as follows:

<TextBox x:Name="InitialFolder" Grid.Row="1" Grid.Column="1"

Grid.ColumnSpan="2" Margin="0,3,2,16"

 Text="{Binding PictureFolders,

Mode=TwoWay, UpdateSourceTrigger=PropertyChanged,

ValidatesOnDataErrors=True, ValidatesOnExceptions=True}"

 ToolTip=

"You may specify more than one full-pathed folder.  Folders are semicolon separated"

ToolTipService.ShowDuration="30000" />

Then the VM, FileTypesToRotateViewModel, in addition to implementing the INotifyPropertyChanged it needs to implement IDataErrorInfo.

I accomplished it through a class, ViewModelBase, that FileTypesToRotateViewModel inherits.

The ViewModelBase implements both INotifyPropertyChanged and IDataErrorInfo. We will concentrate on the IDataErrorInfo. The IDataErrorInfo calls for two (2) properties:

public string this[string propertyName] { get { ... } }

public string Error { get { ... } }

They way the ViewModelBase class implements it is separating the base class from the rules. The base class defines an ErrBinder class and a dictionary of (property name, ErrBinder instance), \_errBinderMap:

private readonly Dictionary<string, ErrBinder> \_errBinderMap = new Dictionary<string, ErrBinder>();

Then a rule agnostic AddRule method:

protected void AddRule(string propertyName, Func<bool> ruleDelegate, string errorMessage)

{

var rv = new ErrBinder(ruleDelegate, errorMessage);

\_errBinderMap.Add(propertyName, rv);

}

That is used by the VM class, FileTypesToRotateViewModel. This VM class knows what the rules should be and as such defines them.

The ErrBinder class’s main function is to allow an error evaluation upon request through the method: ErrEvaluate().

private class ErrBinder

{

private readonly Func<bool> \_ruleValidate;

private readonly string \_message;

internal ErrBinder(Func<bool> ruleValidate, string message)

{

\_ruleValidate = ruleValidate;

\_message = message;

}

internal string Error { get; set; }

internal bool HasError { get; set; }

internal void ErrEvaluate()

{

Error = null;

HasError = false;

try

{

var rc = \_ruleValidate();

HasError = !rc;

if (rc) return;

Error = \_message;

HasError = true;

}

catch (Exception e)

{

Error = e.Message;

HasError = true;

}

}

}

Now filling in the implementation of IDataErrorInfo is easy:

public string this[string propertyName]

{

get

{

if (!\_errBinderMap.ContainsKey(propertyName)) return null;

\_errBinderMap[propertyName].ErrEvaluate();

return \_errBinderMap[propertyName].Error;

}

}

public string Error

{

get

{

var errors = \_errBinderMap.Values.Where(b => b.HasError).Select(b => b.Error);

return string.Join(Environment.NewLine, errors);

}

}

In addition we need a HasErrors property, this method will allow us to set the CanExecute method on the OK button of the form.

protected bool HasErrors

{

get

{

var values = \_errBinderMap.Values.ToList();

values.ForEach(b => b.ErrEvaluate());

return values.Any(b => b.HasError);

}

}

# Points of interest

There are two types of extensions that we filter on: the still pictures extensions and the motion pictures extensions, therefore we need to filter all files based on the union of both sets of extensions. Linq provides a Union() method. This Linq Union() needs to compares objects for equality, so in our extensions case it is sting comparison which by default is case sensitive. We are looking for case insensitive comparison, “.mov” is the same as “.MOV”.

Linq provides a Union() overload method with the following signature:

public static IEnumerable<TSource> [Union](http://referencesource.microsoft.com/#System.Core/System/Linq/Enumerable.cs,d54fdb15d52500f1)<TSource>(this IEnumerable<TSource> first,

IEnumerable<TSource> second, IEqualityComparer<TSource> comparer)

See the site <http://sourceof.net>, where Microsoft publishes the source code to .Net, then search for Enumerable.Union, searching for Union only yields too many hits.

At this point we can provide a specific solution for our problem at hand but what we really would like is a generic solution where we could write the Union() method as follows:

\_stillExt.Union(\_motionExt,

(x, y) => string.Compare(x, y, StringComparison.OrdinalIgnoreCase) == 0);

This is possible by defining our own LinqExtensions static class

public static class LinqExtensions

{

public static IEnumerable<TSource> Union<TSource>(

this IEnumerable<TSource> first,

IEnumerable<T> second,

Func<TSource, TSource, bool> cmpr)

{

// This method calls Linq’s Union

return first.Union(second, new LinqComparer<TSource>(...));

}

}

LinqComparer will be a class that inherits from IEqualityComparer<in T> (see <http://sourceof.net>). The IEqualityComparer<in T> interface demands implementation of:

bool Equals(T x, T y);

int GetHashCode(T obj);

So, for example:

public class LinqComparer<TSource> : IEqualityComparer<TSource>

{

private readonly Func<TSource, TSource, bool> \_linqCmp;

public LinqComparer(Func<TSource, TSource, bool> cmp)

{

\_linqCmp = cmp ?? throw new ArgumentException(

@"comparison function may not be null ", nameof(cmp));

}

public bool Equals(TSource x, TSource y) => \_linqCmp(x, y);

public int GetHashCode(TSource x) => 0;

}

This last GetHashCode(TSource x) => 0 is an issue for a large set, a return of 0 for every GetHashCode() means that a full comparison will be done on every comparison, as opposed to leverage integer comparison via the HashCode.

As such we will include a different approach. LinqComparer will be as follows:

public class LinqComparer<TSource> : IEqualityComparer<TSource>

{

private readonly Func<TSource, TSource, bool> \_linqCmp;

private readonly Func<TSource, int> \_hashCode;

public LinqComparer(Func<TSource, TSource, bool> cmp,

Func<TSource, int> hashCode = null)

{

\_linqCmp = cmp ?? throw new ArgumentException(

@"comparison function may not be null", nameof(cmp));

\_hashCode = hashCode ?? (T => 0);

}

public bool Equals(TSource x, TSource y) => \_linqCmp(x, y);

public int GetHashCode(TSource x) => \_hashCode(x);

}

And the LinqExtensions class will be:

public static class LinqExtensions

{

public static IEnumerable<T> Union<T>(

this IEnumerable<T> a,

IEnumerable<T> b,

Func<T, T, bool> cmpr,

Func<T, int> hashCode = null) =>

a.Union(b, new LinqComparer<T>(cmpr, hashCode));

}

Now calling the Union method is formed as follows:

\_stillExt.Union(\_motionExt,

(x, y) => string.Compare(x, y, StringComparison.OrdinalIgnoreCase) == 0),

x => x.ToLower().GetHashCode());

And LinqComparer is as follows:

public class LinqComparer<TSource> : IEqualityComparer<TSource>

{

private readonly Func<TSource, TSource, bool> \_linqCmp;

private readonly Func<TSource, int> \_hashCode;

public LinqComparer(Func<TSource, TSource, bool> cmp,

Func<TSource, int> hashCode = null)

{

\_linqCmp = cmp ?? throw new ArgumentException(

@"comparison function may not be null", nameof(cmp));

\_hashCode = hashCode ?? (T => 0);

}

public bool Equals(TSource x, TSource y) => \_linqCmp(x, y);

public int GetHashCode(TSource x) => \_hashCode(x);

}

Achieving our goal of using a lambda expression for the comparison of the Union() method.

# Conclusion

We have achieved separation of concerns and a working picture rotation application.