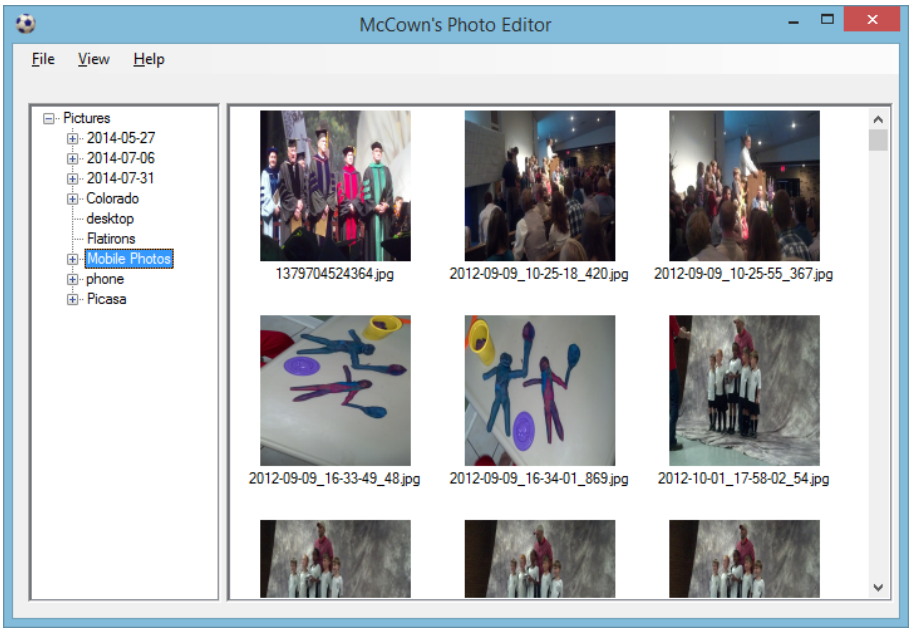
Project 1: **Photo Editor**
GUI Programming
100 Points

Overview

Your team is to create a simple photo editor using C# and Windows Forms. The editor will allow the user to browse the list of available directories and view the JPEG photos within each selected directory. The user can apply various transformations to an image like changing the brightness level, inverting the colors, and making the photo use shades of a particular color. Teams will use git for version control and house their projects in a private repository on github.com.



You will find it helpful to play with my working solution: http://www.harding.edu/fmccown/gui/PhotoEditorMcCown.exe

Requirements

The following specifications must be met in order to receive **100 points**:

1. The main window should use a tree view control on the left and a list view control on the right. The tree view lists the directories the user may browse that contain photos (the user's Pictures directory is the default). The list view will show all the JPEG images in the folder selected from the tree view (no other file formats need to be supported).

2. When the user clicks on a directory from the tree view, the application should load the photos into the list view in a separate thread so the application remains as responsive as possible. Ideally the photos should be displayed in the list view as they are read from disk, but it is OK to display all of them at once after they have completed loading. The user should be able to click on another directory before all the photos from the previously selected directory have loaded. An indeterminate progress bar (marquee) should be displayed while the images are loading into the list view.

3. The program should provide a menu with the following options:
   a. File
      - Locate on Disk: Launches file explorer loaded with the path to the selected image
      - Select Root Folder: Launches the folder browser common dialog box and allows the user to select a new root directory to be displayed in the tree view
      - Exit: Quits the program
   b. View

- Detail: Displays the photo, last modification, and file size (in MB)
- Small: Displays the photo in as a small icon
- Large: Displays the photo as a large icon

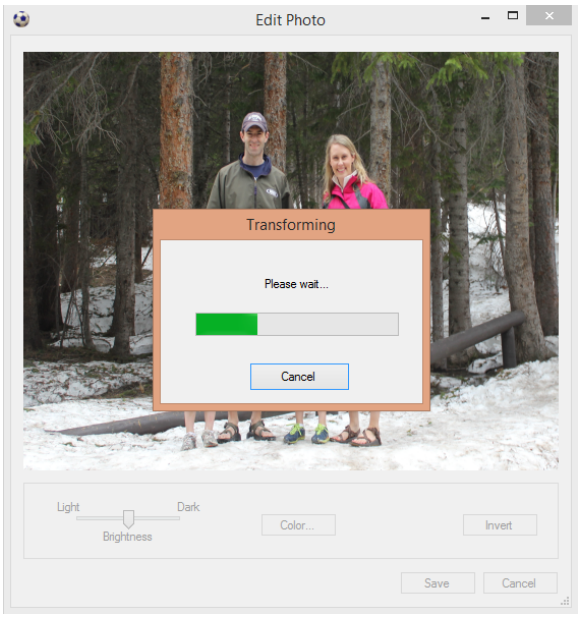
c. Help
- About: Displays an about dialog box listing the developers' names and a summary of what the app does

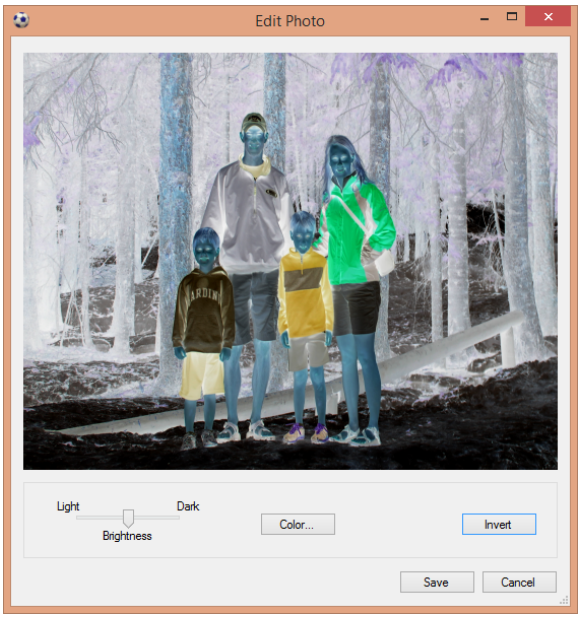
The View menu items should have a checkmark next to the currently selected view mode.

4. The user should be able to double-click on an image in the list view and display the photo in the edit window (a modal window). The window should be resizable so the image stretches or shrinks as the window is resized.

5. The edit window should allow the user to transform the image in a few ways:
   a. Brightness: A track bar can be used to make the image lighter or darker
   b. Color: A button allows the user to choose a color by launching the color common dialog box. The image is then tinted using the selected color
   c. Invert: A button inverts all the colors in the image

   While these operations are being performed, a dialog box with a progress bar should be displayed in the center of the edit window so the user knows how long the operation will take to complete. A Cancel button should be on the dialog box; when pressed it should cancel the current transformation and leave the image just as it was before the transformation began. If the transformation completes without the user pressing Cancel, the edit window should display the newly transformed image.





None of the edit photo window controls should be selectable while the image is being transformed.

6. The edit window should have two control buttons:
   a. Save: saves the current image, replacing the previous image
   b. Cancel: ignores all transformations and does not save the image

   Closing the window via Alt-F4 or the close button should do the same thing as pressing Cancel.

7. The application should have an icon that appears in the Windows task bar and all window title bars.

You may choose to implement any number of these options for extra points:

1. (2 pts) Use a Splitter control to allow the tree view and list view to be resized.
2. (2 pts) Add folder icons to all the folders in the tree list.
3. (3 pts) Show the images in the list view as they are loading.
4. (3 pts) Add a "Save As…" button to the edit window that launches the Save common dialog box so the user can give a new name to the file.
5. (4 pts) Use the JPEG's size to compute the correct aspect ratio to display the image in the edit window. Keep the aspect ratio the same and enlarge or shrink the photo when the edit window is being resized.
6. (5 pts) Add a cropping transformation which allows the user to either select a rectangular portion of the image or to enter values in a dialog box that indicates the left, top, right, and bottom edges to crop.

Teams

Everyone has been assigned to a two-person team:

1. Adam Campbell and C.J. Wilson
2. Cameron Crouch and Alaina Bridges
3. Bryan Cuneo and John Michael Eaton
4. Christian Ellison and Jon-Michael Fields
5. Weicheng (Matthew) Gan and Jeffrey Gasvoda
6. James Goodpasture and Caleb Krise
7. Stephen Hoffmann and Noah Kinslow
8. Garrett Holmes and Anna Crowder
9. Levi Mason and Ricardo Perez
10. Mario Racancoj and Lauren Ragland
11. Jerred Shepherd and Alec Watson
12. Matthew Sorrell and Nicholas Terrill

Both teammates should aim to contribute equally to the project. Teams may use pair programming if desired. Ideally, the project should be implemented in parallel with each teammate responsible for a part of the application.

Each team should create a private repo on github.com and add me (fmccown) as a contributor. Both teammates should consistently push their work to the repository so a history of the project's creation is accurately captured on GitHub.

The repo's README file should contain the following information:

1. Link to the repo on GitHub
2. Brief summary of what the program does
3. List of any known bugs that still remain (hopefully none)
4. List of any extra credit that was implemented
5. List what each teammate contributed (e.g., code to transform images, populate the list view, etc.)
6. The percentage of work performed by each teammate

Ideally the work performed by each teammate will be 50/50, but if one teammate does more than the other, the percentages should be adjusted accordingly. The percentages will influence each teammate's final grade on the project. Note that the repo history on GitHub should be a fairly accurate indicator of how much work each teammate did. Pair programming is one technique that helps to equalize the amount of effort each teammate puts in, but it does not allow for parallel implementation.

Grading

These are the things I will be looking for when evaluating your programs:

1. Does your program implement all the requirements and work properly?
2. Does your program make any GUI bloopers?
3. Is your program well written? Did you use proper C# naming conventions for variables, methods, and classes? Did you use constants and functions when appropriate?

4. Did you clearly cite any code that you may have found online?
5. Did you use git adequately when developing your solution? There should be at **least 20 commits** in your repo, and each person must have committed at least twice. Your .gitignore should keep out any unnecessary files.

Your source code on GitHub should not be modified after you have submitted the README to Easel. I will clone your repository on my machine to run and grade it. One teammate should submit the team's README file to Easel before the deadline; there is no need to submit anything else to Easel.

Note: This is a challenging project to implement. Starting early and getting help when you need it will benefit you greatly!

Note 2: Like any assignment, you are not to share your code with anyone but your teammate! That's why you are using a private repo.

Implementation

To complete some parts of this project, you will need to search online for some example code. Below I have given you some guidance on the trickier parts, but I expect you to do some of your own research. As a professional software developer you will be required to do a lot of independent research, so you need to develop your research skills now. Remember to document where you find any code that you incorporate into your project. Put the URL and the name of the person who wrote the code you are citing in comments.

1. When loading the images from disk into the list view, you will need to use an async method and Task so the UI thread is not locked. Here is some code for traversing a directory and discovering all the JPEG images:

```
DirectoryInfo homeDir = new DirectoryInfo("C:\photos");
foreach (FileInfo file in homeDir.GetFiles("*.jpg"))
{
   try
   {
      byte[] bytes = System.IO.File.ReadAllBytes(file.FullName);
      MemoryStream ms = new MemoryStream(bytes);
      Image img = Image.FromStream(ms);                  // Use this instead of Image.FromFile()
      Console.WriteLine("Filename: " + file.Name);
      Console.WriteLine("Last mod: " + file.LastWriteTime.ToString());
      Console.WriteLine("File size: " + file.Length);
   }
   catch
   {
      Console.WriteLine("This is not an image file");
   }
}
```

   Note that you will probably want to store each file into a List of `FileInfo` objects so you can create `ListViewItems` for each file that holds the image's filename, last modification date, and size. The `ListViewItems` would be added to your list view (with the proper index) in order to display the images. You should not hard-code the path to the user's images but instead look in the user's Pictures directory by default (google to find out how).

2. When the user clicks on another directory in the tree view before the current Task is finished reading the images, the Task must be cancelled and restarted so the Task can read the images from the newly selected directory. Here's one way of waiting until the Tak has completed:

```
// Cancel the task (assume the token is not null when task is running)
if (cancellationTokenSource != null)
{
   cancellationTokenSource.Cancel();

   // Wait until the task has been cancelled
   while (cancellationTokenSource != null)
   {
      Application.DoEvents();
   }
```

```
}
```

3. In order to populate the tree view controller with the proper directory structure, take a look at this solution: http://stackoverflow.com/questions/6239544/c-sharp-how-to-populate-treeview-with-file-system-directory-structure

4. To determine if the user has selected (double-clicked) a photo in the list view, create an ItemActive event listener. You should probably set the list view's MultiSelect property to false so the user can only select one image at a time.

5. To save a modified image back to disk, use the Image's Save method:

```
myImage.Save("myohoto.jpg", ImageFormat.Jpeg);
```

   You may want to make the saved image into a public property of the edit photo window so the main form can access the image.

6. You will need to use another async method and Task for performing the image transformations. The transformation dialog box which shows the progress bar should be implemented as a modeless dialog box so the edit photo window is not blocked while the transformation dialog box is displayed. You will need to register a callback with the modeless dialog box so you can be alerted if the user presses the Cancel button and handle it appropriately. You will also need to create a public property for the dialog box so it can be told about the progress the Task is making in the edit photo window and update its progress bar. This article might be helpful to you: http://perschluter.com/show-progress-dialog-during-long-process-c-sharp/

   Your logic will look something like this:

```
// User moves the track bar or presses a button to start a transformation
Launch the progress dialog box
Disable the edit form
transformedBitmap = pictureBox's image

If selectedTransformation = Invert then start Task to invert image
Else if selectedTransformation = ChangeColor then start Task to change colors
Else if selectedTransformation = ChangeBrightness then start Task to change brightness

Close the progress dialog box
Enable the edit form and bring it to the front
If the Task wasn't cancelled then pictureBox's image = transformedBitmap
```

7. In order to implement the image transformations, you will need to loop through each pixel of the Bitmap that represents the JPEG. The following code implements the Invert transformation. You would modify this slightly to implement the other transformations.

   Please note that this function is missing the logic to handle cancelling and notifying the transformation progress dialog box about the progress it is making. When reporting your progress, you will only want to report it when you have made at least 1% progress, *not* for every single pixel which will slow the operation down significantly.

```
private void InvertColors()
{
   for (int y = 0; y < transformedBitmap.Height; y++)
   {
      for (int x = 0; x < transformedBitmap.Width; x++)
      {
         Color color = transformedBitmap.GetPixel(x, y);
         int newRed = Math.Abs(color.R - 255);
         int newGreen = Math.Abs(color.G - 255);
         int newBlue = Math.Abs(color.B - 255);
         Color newColor = Color.FromArgb(newRed, newGreen, newBlue);
         transformedBitmap.SetPixel(x, y, newColor);
      }
   }
}
```

Note: There are other more efficient ways to perform the image transformations that you might discover online, but I do not want you to use them; the purpose of this part of the project is to give you experience with performing work in a background thread and updating a progress bar in the UI thread.

8. To implement the color transformation, first calculate the average of the RGB values for a pixel. This will be a number between 0 and 255. Then convert this number to a percentage (divide by 255). Multiply the RGB values of the selected transformation color by the percentage to get the new RGB values for the pixel.

   For example, if the RGB values of the pixel were (255, 100, 0), the average would be (255 + 100 + 0)/3 = 118.3. Converted into a percent, it would be 118.3/255 = 46.4%. If the user had select blue (0, 255, 0) as the transformation color, the pixel's new RGB values would be (0 * 46.4%, 255 * 46.4%, 0 * 46.4%) = (0, 118, 0).

9. For the brightness track bar, start the value in the middle of the track bar's range. So if the minimum is 0 and the maximum is 100, the middle is 50. The middle value will represent no change. When the bar is moved to the left, we want to make the image darker; when it is moved to the right, we want the image to be lighter. You can calculate the amount to subtract from each pixel's RGB values using the following equation:

   ```
   int amount = Convert.ToInt32(2 * (50 - brightnessTrackBar.Value) * 0.01 * 255);
   ```

   So if the track bar was moved all the way to the right, amount would be 2 * (50 – 100) * 0.01 * 255 = -255. So we would subtract -255 from each pixel's RGB values which would of course *add* 255 to each value. That could make some colors have an RGB value larger than 255, so you would need to cap off each value at 255. You would also need to make sure no RGB value was smaller than 0.