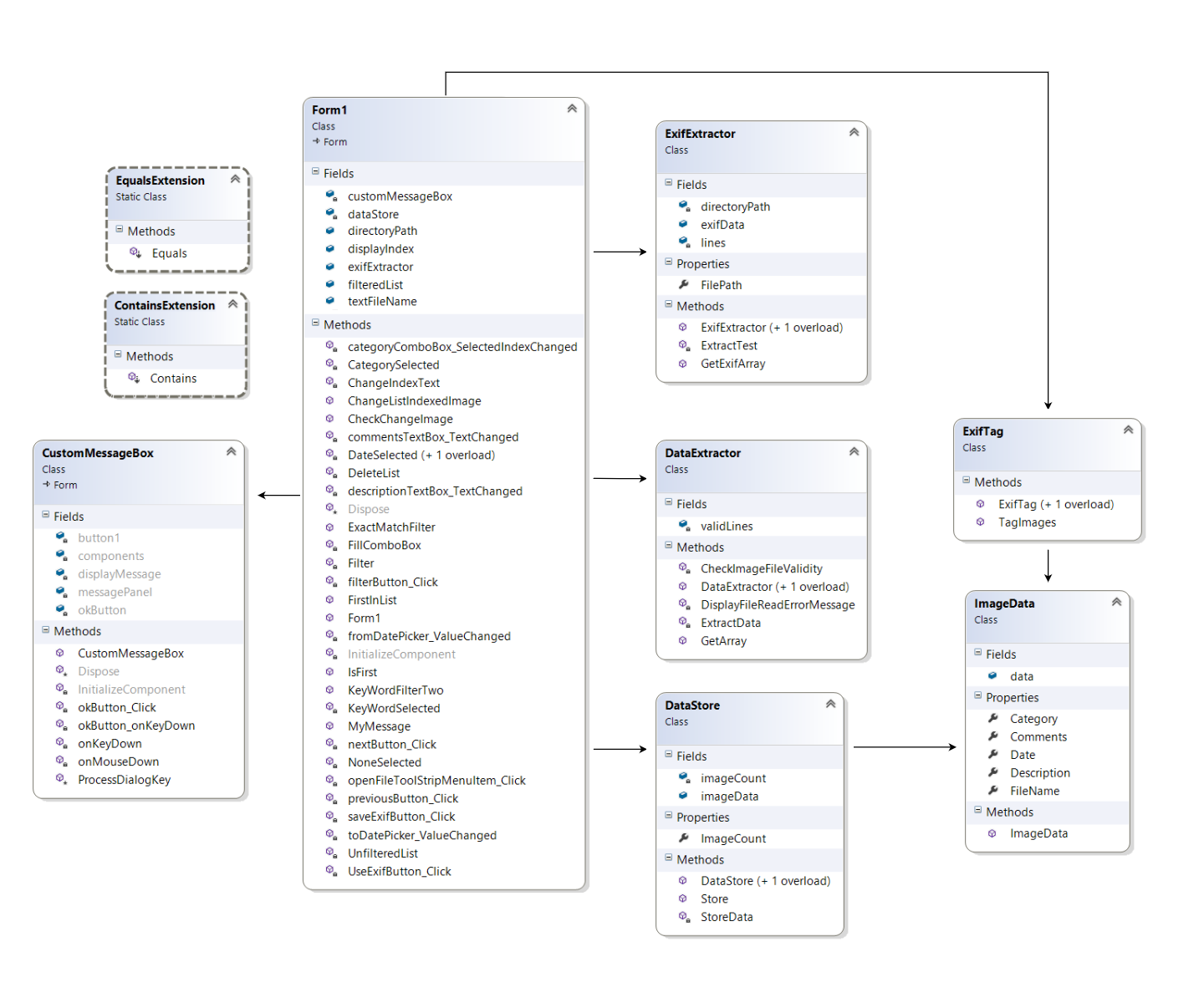
Picture Management

# Contents

1. Main Body
   * 1. Classes methods and their properties
     2. Data Structures
     3. Text file structure
     4. Enhancements
2. Critical Evaluation
   * 1. Critique
     2. Hindsight

Main Body

i. As this is a WinForms application, code dealing with user interactions, input and controls resides in the Form1 partial class. The Form1 class implements a has-a relationship with all other classes in the project besides static classes and ImageData(). The ChangeListIndexedImage() changes the displayed image. A second form MyMessage() displays a simple message which can be dismissed by pressing any key or clicking the mouse regardless of location. Other classes used are the DataExtractor() class for extracting tags and image data. DataStore() takes an array as an argument and stores the image tags in an array of class ImageData(), the ralationship of these classes is a has-a relationship. ImageData() contains an array which stores the tags of an image and fields so they can be requested and updated with ease. Two static classes extending the .Contains and the .Equals methods respectively, allowing for case insensitive comparisons of strings. The methods in Form1 ExactMatchFilter() and KeyWordFilter() call make use of these static classes. Additional classes have been created to handle Exif data.



ii. A List<T> of strings is used to store file names of the image displayed and images that can presently be indexed through. This list is cleared when filter is applied, if an image meeting the filter criteria is identified. This image and subsequent matches then repopulate the list. The use of a list populated from information stored in an array allows filters to be applied without repeatedly reading the .txt file. Other data types used include various arrays mostly for storing batches of file names or tag data extracted from text files or Exif data.

iii. The text file format consists of five tagIDs consisting of the initials of the tag IDs FN=, DT=,CT=, DS=, CM=,> the relevant data directly follows as shown below:

FN=Beach.jpg,DT=2017-06-20,CT=Holiday,DS=Sea and sky,CM=Amazing,>FN= FileName,DT=Data,CT=Category, DS=Description, CM=Comments,>//

Each tag is separated from the next with a comma and each set of tags ends with an angle bracket. Each tag must directly follow the next, any spaces or other characters before or instead of the tagID will result in an unread tag, missing commas will merge tags. Using a tagID requires an extra step to check but the advantage is that a missing tag or comma will not disrupt every following tag. The end of the information is denoted with two forward slashes “//” placed directly after the last tag. Instructions and comment can be written after this point without interfering with the program. The date format conforms to ISO 8601 Date and time format standard YYYY-MM-DD as data stored in the Exif tag used should conform to this standard.

iv. The program has extra functionality providing the ability to read and write Exif data to and from images. This is achieved through two additional classes, ExifTag and ExifExtract. ExifTag saves tags from the .txt file to new copies of the images as Exif data, which are stored in a folder of the users choice. The class ExifExtractor can Exif data previously saved to images by this or other applications allowing for such images to be displayed and filtered without the use of a text file. ExifExtractor formats the data it reads to structuring it to the same parameters as tags in our text file and passes it in an array of lines. The exif data is stored in appropriate Exif tag locations and as such tags can be viewed in Windows File Explorer for example.

Evaluation

i. In my view the Picture Management application successfully meets the required specifications, plus carries out the complex tasks of reading and writing Exif tags. There are of course improvements that could be made, the code could be more modular in areas, there are some known bugs and almost certainly more to be fixed.

There is a bug resulting from the fact that non .txt files may be loaded by the user as text files. The program reads the file, displaying messages informing the user that no relevant tags can be found, until the program has run its course. This could be resolved by checking the file extension. Another error is triggered when the user opens a file dialog, then cancels without selecting a file. If the programs open file dialogs were handled by a single class this would be far easier to fully fix. Currently this potential lack of input is dealt with in some areas, but not others. Over all I feel the error handling and input validation whilst imperfect is for the most part decent.

In the data extractor Class, one flaw is that invalid files are simply ignored and not added to an array. The user should be informed that a file has been determined invalid. Also the present means of excluding invalid files simply by checking their extension could be improved. The DataExtractor class is quite verbose and complex, the CheckImageFileValidity method should perhaps be a class of its own.

ii.Multiple uses of the open file dialog were not foreseen, with hindsight creating a class OpenFile to handle all open file dialogs, would have been the correct approach. Doing so removes repetition and better conforms to object-oriented principles. In theory this would allow for the many potential errors to be prevented in one place with greater ease. Presently the project has been reverted to a prior version due to difficulties and mistakes made whilst attempting to implement this. With regards to the Form1 partial class, though many methods are linked to UI controls, more methods could potentially be classes of their own. If these changes were implemented and DataExtractor altered the class structure would be improved.