Bitmap Tagger: An Open Source Image Tagging System

Author: Christopher Treml

Faculty Advisor: Dr. Kenny Hunt, Department of Computer Science

Abstract

This paper describes an open source software system for reading and writing tags in images. The software system consists of a library and a GUI application for viewing and editing tags and supports both JPEG and PNG file formats. The software system can read and write common tags for the Exif and XMP specifications.

Project Goals

The long term goal of this project is to build an easy to use open source .NET library that can read and write a variety of image tags for a variety of image types. The ideal library would be able to read and write the most commonly used tags in the Exif and XMP specifications. The BitmapTagger Library, referred to throughout this paper as the library, would be able to read and write these tags in JPEG and PNG image formats, with more image formats supported in future versions. The library would abstract away the intricate details of the tagging specifications and image formats. This tagging interface would present the various tags as .NET properties, allowing software developers to read and write tags in the same manner they would read or write a property belonging to any other .NET class.

Another goal of this project is to build a GUI application based upon the library that end users could use to view and edit tag information in images. The GUI would serve as a sample application and demonstrate how other developers could include the core library into custom software solutions. The GUI application’s standard user profile would be an average PC user, meaning no technical knowledge would be required.
Project Description

The library created in this project uses a Microsoft technology known as the Windows Imaging Component (WIC) [3]. WIC is a technology accessible in .NET that provides a uniform method for accessing image data and image metadata. WIC was introduced in Windows Vista and Windows XP SP3. In .NET terms, WIC is analogous to the SQLConnector class. WIC abstracts away details such as how information is sent and obtained, but still requires the user to know the details of the data they are working with. The library in this project provides another layer of abstraction on top of WIC that handles details such as how to store and retrieve tag information and data conversion. The library itself is also a large piece of documentation on how to use WIC, which was done by keeping the library well documented and open sourced. The strong documentation of the library provides other developers using WIC an excellent resource.

The library is designed to provide uniform access to the Exif and XMP tag specifications. The Exif specification is a metadata specification developed by the Japan Electronic Industries Development Association (JEIDA). Exif is a binary based format and can only be used in JPEG and TIFF images[1]. The XMP specification is a metadata specification developed by Adobe. XMP is an XML based format that is can be used in a large variety of file formats including PNG, JPEG, and PDF [2]. XMP supports a variety of schemas that allow it to be very extensible. For instance, XMP supports many of the Exif tags in its Exif schema. Throughout this project the Exif schema in XMP is referred to as XMPExif.

The library provides uniform access to tag specifications by using an abstract class called BitmapTagger that contains the tags as properties. The BitmapTagger’s tag properties are the tags common to all its child classes. Each child class of BitmapTagger is designed to support one type of image format. Each child class supports only one image format because the methods to read and write the tag information varies for each image format. By having a design based on the use of a central abstract class the library also allows new image formats to be supported in future versions. BitmapTagger also contains a static method called Create that returns an object of type BitmapTagger. The Create method implements a variation of the Factory design pattern that is commonly used throughout the .NET framework. Using the Create method to construct the correct BitmapTagger child class allows a user of the library to easily create a BitmapTagger object without having to know details such as which child classes support what formats.

Since the library supports multiple tag specifications it is possible to have semantic conflicts in the image tags. Conflicts could occur when two similar tags, such as ImageDescription in Exif and dc:description in XMP, contained different values. To decide how to solve these conflicts the user of the library can set the precedence of the tag specification.
The value of the precedence decides which tag specification values are used when a conflict occurs. The current BitmapTagger properties and the exact tags that correspond to them can be seen in Table 1.

<table>
<thead>
<tr>
<th>BitmapTagger Property</th>
<th>Exif</th>
<th>XMPExif</th>
<th>XMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>XResolution</td>
<td>XResolution</td>
<td>tiff:XResolution</td>
<td>N/A</td>
</tr>
<tr>
<td>YResolution</td>
<td>YResolution</td>
<td>tiff:YResolution</td>
<td>N/A</td>
</tr>
<tr>
<td>ResolutionUnit</td>
<td>ResolutionUnit</td>
<td>tiff:ResolutionUnit</td>
<td>N/A</td>
</tr>
<tr>
<td>DateTime</td>
<td>DateTime</td>
<td>tiff:DateTime</td>
<td>xmp:ModifyDate</td>
</tr>
<tr>
<td>ImageDescription</td>
<td>ImageDescription</td>
<td>tiff:ImageDescription</td>
<td>dc:description</td>
</tr>
<tr>
<td>Make</td>
<td>Make</td>
<td>tiff:Make</td>
<td>N/A</td>
</tr>
<tr>
<td>Model</td>
<td>Model</td>
<td>tiff:Model</td>
<td>N/A</td>
</tr>
<tr>
<td>Software</td>
<td>Software</td>
<td>tiff:Software</td>
<td>xmp:CreatorTool</td>
</tr>
<tr>
<td>Artist</td>
<td>Artist</td>
<td>tiff:Artist</td>
<td>dc:creator</td>
</tr>
<tr>
<td>Copywrite</td>
<td>Copywrite</td>
<td>tiff:Copyright</td>
<td>dc:rights</td>
</tr>
<tr>
<td>UserComment</td>
<td>UserComment</td>
<td>exif:UserComment</td>
<td>N/A</td>
</tr>
<tr>
<td>DateTimeOriginal</td>
<td>DateTimeOriginal</td>
<td>exif:DateTimeOriginal</td>
<td>N/A</td>
</tr>
<tr>
<td>DateTimeDigitized</td>
<td>DateTimeDigitized</td>
<td>exif:DateTimeDigitized</td>
<td>N/A</td>
</tr>
<tr>
<td>MakerNote</td>
<td>MakerNote</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>MeteringMode</td>
<td>MeteringMode</td>
<td>exif:MeteringMode</td>
<td>N/A</td>
</tr>
<tr>
<td>SensingMethod</td>
<td>SensingMethod</td>
<td>exif:SensingMethod</td>
<td>N/A</td>
</tr>
<tr>
<td>ExposureProgram</td>
<td>ExposureProgram</td>
<td>exif:ExposureProgram</td>
<td>N/A</td>
</tr>
<tr>
<td>ExposureTime</td>
<td>ExposureTime</td>
<td>exif:ExposureTime</td>
<td>N/A</td>
</tr>
<tr>
<td>ISOSpeedRatings</td>
<td>ISOSpeedRatings</td>
<td>exif:ISOSpeedRatings</td>
<td>N/A</td>
</tr>
<tr>
<td>LightSource</td>
<td>LightSource</td>
<td>exif:LightSource</td>
<td>N/A</td>
</tr>
<tr>
<td>FNumber</td>
<td>FNumber</td>
<td>exif:FNumber</td>
<td>N/A</td>
</tr>
<tr>
<td>ShutterSpeed</td>
<td>ShutterSpeedValue</td>
<td>exif:ShutterSpeedValue</td>
<td>N/A</td>
</tr>
<tr>
<td>Brightness</td>
<td>BrightnessValue</td>
<td>exif:BrightnessValue</td>
<td>N/A</td>
</tr>
<tr>
<td>Aperture</td>
<td>ApertureValue</td>
<td>exif:ApertureValue</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 1. BitmapTagger Properties and Corresponding Tag Values

An example of a possible conflict would be the ImageDescription. From Table 1 we can see there are three different tags that could contain a value for this property. If the Exif, XMPExif, and XMP tags all contained different values the tag precedence would be used to determine which tag’s value would be used as the value for ImageDescription.
Figure 1: High Level Class Diagram of BitmapTagger Library

Figure 1 gives a UML class diagram of the core classes and does not show the many helper classes used in the library. BitmapTagger contains methods needed by all child classes for normalizing data, such as the GetDateTimeFromExifString and GetDateTimeFromXMPString methods. The Load and Save methods in BitmapTagger are marked as abstract and implemented in JpegTagger and PngTagger. Not shown in Figure 1 are the properties contained in BitmapTagger. These properties are the image tags. Every tag in BitmapTagger is a nullable value type, this was necessary for two reasons: all tags are not present in all images and setting a tag to null allows a user to remove a tag.

BitmapTagger Library User’s Guide
To use the library in a .NET project the BitmapTagger DLL must first be referenced in the project. The namespace for the library is “BitmapTagger”. There are a few ways to load an image once the library is referenced, as can be seen in Figure 2.

```csharp
string jpegLocation = @"C:\Users\Chris\Desktop\TaggerTests\mytest.jpg";
string pngLocation = @"C:\Users\Chris\Desktop\TaggerTests\mytest.png";

//Load image with default values: load all tag specifications,
//precedence Exif > XMPExif > XMP
var tagger = BitmapTagger.BitmapTagger.Create(jpegLocation);

//Load image with specified tag specifications and tag precedence
tagger = BitmapTagger.BitmapTagger.Create(jpegLocation,
true, false, false, TagPrecedence.XMP_XMPExif_Exif);

//Load png image directly with png tagger class
//Load with default values: load all tag specifications,
//precedence: Exif > XMPExif > XMP
var pngTagger = new PngTagger(pngLocation);

//Load png image directly with png tagger class
//Load with specified tag specifications and tag precedence
pngTagger = new PngTagger(pngLocation, true, false,
TagPrecedence.XMP_Exif_XMPExif);

//Load jpeg image directly with jpeg tagger class
//Load with default values: load all tag specifications,
//precedence: Exif > XMPExif > XMP
var jpegTagger = new JpegTagger(jpegLocation);

//Load jpeg image directly with jpeg tagger class
//Load with specified tag specifications and tag precedence
jpegTagger = new JpegTagger(jpegLocation, true, false, false,
TagPrecedence.XMP_XMPExif_Exif);

Figure 2. Load Image Code

As seen in Figure 2 the parameters to load an image consist of an image location, Booleans, and a TagPrecedence. The Booleans are used to decide which tag specifications to load and the TagPrecedence is used to decide the precedence to use in cases of conflicts. Alternatively the user can just use default values instead of specifying the tag specifications loaded and the tag precedence. The default values when loading an image are all tag specifications are loaded and the tag precedence is set to Exif > XMPExif > XMP.

```csharp
string imageLocation = @"C:\Users\Chris\Desktop\TaggerTests\mytest.jpg";

var tagger = BitmapTagger.BitmapTagger.Create(imageLocation);

//If the Image Description information was read print
// to console
if (tagger.ImageDescription != null)
    Console.WriteLine("Image Description is: {0}",
                      tagger.ImageDescription);

// Change the Image Description to a new value
tagger.ImageDescription = "My New Description";

// If the DateTimeDigitized exists delete it
if (tagger.DateTimeDigitized != null)
    tagger.DateTimeDigitized = null;

Figure 3. Read and Edit Tag Information

Figure 3 shows a code fragment that reads the ImageDescription and DateTimeDigitized tags from an image. The code then deletes the DateTimeDigitized tag if it existed. Once an image is loaded its tag information can be directly read and written using the properties in the BitmapTagger class. A value of null means the tag was not present in the image and setting a tag to a value of null deletes the tag.

string imageLocation = @"C:\Users\Chris\Desktop\TaggerTests\mytest.jpg";

var tagger = BitmapTagger.BitmapTagger.Create(imageLocation);

// Delete DateTimeDigitized if it exists
if (tagger.DateTimeDigitized != null)
    tagger.DateTimeDigitized = null;

// Set the Image Description
tagger.ImageDescription = "Hello World";

// Save directly back to the image's original location
// with the tag information saved to all tag specifications
tagger.Save();

// Save the image with the new tag information to a different location from the one it was loaded from
// with the tag information saved to all tag specifications
tagger.Save(@"C:\Users\Chris\Desktop\TaggerTests\mytest1.jpg");

// Save back to the image's original location
// only saving the tag information to the specified
// tag specifications
tagger.Save(true, false, false);

// Save the image with the new tag information to a different location only saving teh tag infromation to the specified tag specifications
tagger.Save(@"C:\Users\Chris\Desktop\TaggerTests\mytest1.jpg", true, true, false);
Figure 4: Save Tag Information

As seen in Figure 4, programmers using the library have several ways to save images. Users can save the image to the location it was loaded from or provide a new location the image should be saved to. The user can also specify which tag specifications the tag information should be saved in or use the default value of all tag specifications. While not shown in Figure 4, an important limitation for developers to know is that any thread that calls the library’s Save methods must be marked as a STAThread (Single-Single Threaded Apartment). This is due to a limitation cause by WIC[3], this should not normally cause a problem as all WPF, WinForms, and console applications are by default marked as STAThread.

ImageTagEditor User’s Guide

ImageTagEditor is the GUI application developed on top of the library aimed at end users. As mentioned before ImageTagEditor allows user to view and modify image tag information. A screen shot of ImageTagEditor can be seen in Figure 5.
Figure 5. ImageTagEditor Main Screen

As seen in Figure 5, the ImageTagEditor shows one image at a time. The main screen is broken into four parts: the tag information (such as Exif data generated from a digital camera) seen on the right side of the screen, the image being viewed seen on the left side of the screen, the menus seen on the top of the screen, and the image selection seen on the bottom of the screen. The tag information is grouped into four categories: General, Description, DateTime, and Picture Conditions.

The information in the general category is derived from the actual image. The general category refers to information such as location, height, and width. The information found in the general category is not editable since this is not metadata, but actual image data.

The information contained in the description, date time, and picture conditions categories is editable. The description category contains information such as image description, artist, copyright, and make. The date time category contains information such as the date and time the image was taken and the date and time the image was digitized. The picture conditions category contains information that pertains to the camera settings used when the image was taken such as FNumber and ISO Speed Ratings.
ImageTagEditor can open a single image or a folder of images. Opening a single image or a folder of images can be done through the File Menu, dragging and dropping files/folders onto ImageTagEditor, or with keyboard shortcuts (which will be discussed later). Saving images can also be done through the File Menu or keyboard shortcuts. The image being viewed can be closed through the File Menu or keyboard shortcuts. Changing the image being viewed is done by clicking the left or right arrows near the bottom of the screen, the mouse scroll wheel, or keyboard shortcuts.

Clicking the Options menu will show the Options Dialog as seen in Figure 6.

![Figure 6: Options Menu](image)

As seen in Figure 6, the options menu allows the user to change the application’s settings such as what tag specifications are loaded, what tag specifications are saved, tag precedence, and what tags to show or hide.

When the ImageTagEditor application is loaded it will load the settings used when it was last closed. These settings include all options available in the options menu (Figure 6), the images opened, the window height, the window width, and what categories were expanded or collapsed.

As mentioned before ImageTagEditor has several keyboard shortcuts, Table 2 explains these shortcuts.

<table>
<thead>
<tr>
<th>Shortcut Key Combination</th>
<th>Action</th>
</tr>
</thead>
</table>

---
Chrisopher Treml

<table>
<thead>
<tr>
<th>Ctrl + S</th>
<th>Save the image being viewed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl + Shift + S</td>
<td>Save all opened images</td>
</tr>
<tr>
<td>Ctrl + Alt + S</td>
<td>Save the image being viewed to a new location</td>
</tr>
<tr>
<td>Ctrl + O</td>
<td>Open a new image</td>
</tr>
<tr>
<td>Ctrl + F</td>
<td>Open a folder of images</td>
</tr>
<tr>
<td>Ctrl + W</td>
<td>Close currently image being viewed</td>
</tr>
<tr>
<td>Ctrl + Right Arrow</td>
<td>Show the next opened image</td>
</tr>
<tr>
<td>Ctrl + Up Arrow</td>
<td>Show the next opened image</td>
</tr>
<tr>
<td>Ctrl + Left Arrow</td>
<td>Show the previous opened image</td>
</tr>
<tr>
<td>Ctrl + Down Arrow</td>
<td>Show the previous opened image</td>
</tr>
</tbody>
</table>

Table 2. ImageTagEditor’s Keyboard Shortcuts

Future Work

This project has several ways it can be improved in the future. The most pressing issue is that currently tags are not properly stored and read from PNG images. Tags are not stored correctly in PNGs because XMP data must be stored in an itxt chunk in PNG files. The current version of WIC does not support reading or writing chunks of type itxt. The library instead reads and writes XMP tags in PNGs from a txt chunk, meaning XMP data is not being stored correctly in PNG images. The library will not destroy or overwrite any XMP in PNG files; the library is just reading and writing from a different chunk in PNG altogether. The next version of WIC, due to be released at the same time as Windows 7, will support reading and writing itxt chunks in PNGs. Once the new version of WIC is released BitmapTagger should be updated to read and write from itxt instead of txt to fully support XMP in PNGs.

The library can be expanded to read and write more Exif and XMP tags such as GPS information. Adding support for more tags is done by adding the tags as a property to the BitmapTagger class and then implementing how the tags are written and read in each child class of BitmapTagger. The library can also be expanded to support more image formats such as TIFF or HD Photo. Adding support for new image formats can be done by creating a new child class from BitmapTagger that supports the specific image format.

The ImageTagEditor could have its options and settings expanded to allow users to have a greater variety of customization such as a larger choice of shown or hidden tags. The ImageTagEditor could also have its image loading functionality changed to load asynchronously to help the user experience when loading many large images. Adding asynchronous support to the ImageTagEditor can be done by changing the method used to show the image to make use of threads to perform the loading of the image data.
Where to Obtain BitmapTagger

BitmapTagger is distributed freely under the BSD license. Source code and releases can be found at the following URL: http://bitmaptagger.codeplex.com/. All questions and issues regarding BitmapTagger are also discussed at the same URL.

References