using ImageMagick;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Web;

namespace Ben.Classes

{

/// <summary>

/// A class that takes a png/gif/jpg and automatically saves it to the only filetype you'll ever need - jpeg.

/// A wrapper class for ImageMagick that gets the important stuff

/// from an image file's metadata so that you can do neat sh\*t with it!

/// </summary>

/// <remarks> free to use - distributed under the [MIT License](Learn more at http://opensource.org/licenses/mit-license.php):

/// Requires Ben.Classes.ExtensionHelpers

/// </remarks>

/// <see>https://www.bennysutton.com/</see>

/// <seealso cref=“https://github.com/BennySutton/”/>

public class Jpeg : IDisposable

{

private MagickImage \_image;

private string \_keywords;

/// <summary>

/// load from an image file on your hard disk

/// </summary>

/// <param name=“path”>root relative virtual path to the script or path for the current request e.g. ~/admin/imagename.jpg or /admin/imagename.jpg </param>

public Jpeg(string rootpath)

{

// prevent accidental reloading on top of existing

if (\_image != null) throw new Exception(“Image already loaded”);

if (!rootpath.GetFileName().IsImage()) throw new Exception(“File does not have an image file extension”);

if (!File.Exists(rootpath.PhysicalPathFromRootPath())) throw new Exception(“The file specified to load does not exist”);

Source = rootpath.GetFileName();

\_image = new MagickImage(rootpath.PhysicalPathFromRootPath());

Init();

}

/// <summary>

/// load from another MagickImage image

/// </summary>

/// <param name=“image”></param>

public Jpeg(MagickImage image)

{

// prevent accidental reloading on top of existing

if (\_image != null) throw new Exception(“Image already loaded”);

if (!image.FileName.IsImage()) throw new Exception(“File does not have an image file extension”);

Source = image.FileName;

\_image = image;

Init();

}

/// <summary>

/// load an image (png/gif/jpg format) from a file upload

/// </summary>

/// <param name=“httpPostedFile”>an image file uploaded from a client</param>

public Jpeg(HttpPostedFileBase httpPostedFile)

{

// prevent accidental reloading on top of existing

if (\_image != null) throw new Exception(“Image already loaded”);

if (!httpPostedFile.FileName.GetFileName().IsImage()) throw new Exception(“File does not have an image file extension”);

try

{

Source = httpPostedFile.FileName.GetFileName();

\_image = new MagickImage(httpPostedFile.InputStream);

Init();

}

catch

{

throw;

}

}

/// <summary>

/// persist existing metadata values (if any) to property values

/// </summary>

private void Init()

{

IExifProfile exifprofile = \_image.GetExifProfile();

if (exifprofile != null)

{

if (exifprofile.GetValue(ExifTag.Copyright) != null) { Copyright = exifprofile.GetValue(ExifTag.Copyright).ToString(); }

if (exifprofile.GetValue(ExifTag.Artist) != null) { Creator = exifprofile.GetValue(ExifTag.Artist).ToString(); }

if (exifprofile.GetValue(ExifTag.ImageDescription) != null) { Subject = exifprofile.GetValue(ExifTag.ImageDescription).ToString(); }

if (exifprofile.GetValue(ExifTag.Software) != null) { Software = exifprofile.GetValue(ExifTag.Software).ToString(); }

}

IIptcProfile iptcprofile = \_image.GetIptcProfile();

if (iptcprofile != null)

{

if (iptcprofile.GetValue(IptcTag.Country) != null) { Country = iptcprofile.GetValue(IptcTag.Country).ToString(); }

if (iptcprofile.GetValue(IptcTag.Headline) != null) { Headline = iptcprofile.GetValue(IptcTag.Headline).ToString(); }

if (iptcprofile.GetValue(IptcTag.Keyword) != null) { Keywords = iptcprofile.GetValue(IptcTag.Keyword).ToString(); }

if (iptcprofile.GetValue(IptcTag.Source) != null) { Source = iptcprofile.GetValue(IptcTag.Source).ToString(); }

if (iptcprofile.GetValue(IptcTag.Caption) != null) { Subject = iptcprofile.GetValue(IptcTag.Caption).ToString(); }

if (iptcprofile.GetValue(IptcTag.Title) != null) { Title = iptcprofile.GetValue(IptcTag.Title).ToString(); }

}

}

public bool Save(string rootpath)

{

bool success = false;

ImageMagick.ExifProfile exifprofile = new ImageMagick.ExifProfile();

exifprofile.SetValue(ExifTag.Copyright, “ ©” + Copyright);

exifprofile.SetValue(ExifTag.Artist, Creator);

exifprofile.SetValue(ExifTag.ImageDescription, Subject);

exifprofile.SetValue(ExifTag.Software, Software);

\_image.AddProfile(exifprofile);

ImageMagick.IptcProfile iptcprofile = new ImageMagick.IptcProfile();

iptcprofile.SetValue(IptcTag.CopyrightNotice, “No Unauthorized reproduction ©” + Copyright);

iptcprofile.SetValue(IptcTag.Byline, Creator);

iptcprofile.SetValue(IptcTag.Country, Country);

iptcprofile.SetValue(IptcTag.Headline, Headline);

iptcprofile.SetValue(IptcTag.Keyword, Keywords);

iptcprofile.SetValue(IptcTag.Source, Source);

iptcprofile.SetValue(IptcTag.Caption, Subject);

iptcprofile.SetValue(IptcTag.Title, Title);

\_image.AddProfile(iptcprofile);

\_image.Write(rootpath);

success = true;

return success;

}

/// <summary>

/// Save the image to a JPEG

/// </summary>

/// <param name=“resizeTo”>Optionally resize the image to a value between 100 and 3000 pixels (longest side)</param>

/// <param name=“rootpathToSaveTo”>the virtual path on the server to save to e.g. “~/images/” </param>

/// <param name=“fileName”>the name of the file that you want to save to (including “.jpg” extension)</param>

/// <param name=“overWrite”>allow/stop overwriting of existing file</param>

/// <remarks>requires Ben.ExtensionHelpers</remarks>

public void ResizeAndSave(string rootpathToSaveTo, string fileName, int resizeTo = 0, bool overWrite = true)

{

// validation of parameters passed in and check path related constraints

// bit of a code smell here but validation code below only happens in this method so do not refactor for now...

string physicalSaveToFolder = rootpathToSaveTo.PhysicalPathFromRootPath(); // convert virtual website path to server file system path

if (!Directory.Exists(physicalSaveToFolder)) throw new Exception(“The directory specified to save to does not exist”);

fileName = string.Join(““, fileName.Split(Path.GetInvalidFileNameChars())); // remove any accidentally passed in characters that are illegal in a filename e.g. / \ [ ] : ; | , ?

Path.ChangeExtension(fileName, “jpg”); // ensure the filename is jpeg compliant - needed for the automatic encoder later

string filepath = Path.Combine(physicalSaveToFolder, fileName);

if (overWrite = false && File.Exists(filepath)) throw new Exception(“This would overwrite an existing file”); // to stop overwriting of existing file if parameter overwite = false

// resize image to max number of pixels longest side

// limit minimum and maximum file dimensions

if (resizeTo >= 1 && !Enumerable.Range(100, 3000).Contains(resizeTo)) throw new Exception(“The file size specified to save to must be between 100px and 3000px”);

var size = \_image.Height <= \_image.Width

? new MagickGeometry(resizeTo, 0)

: new MagickGeometry(0, resizeTo);

size.IgnoreAspectRatio = false;

MagickImage clone = new MagickImage(\_image);

clone.Resize(size);

clone.Write(filepath);

}

/// <summary>

/// resets Keywords - when you add keywords with multiple calls to Keyword they are preserved so this is the only way to reset them - don't forget to call Save() afterwards

/// </summary>

public void ClearKeywords()

{

\_keywords = ““;

}

/// <summary>

/// Allows you to loop through all the fields e.g. on a razor page

/// </summary>

/// <example>Ben.Classes.Jpeg magickMetadata = new Ben.Classes.Jpeg(filename);

/// Dictionary<string, object> myDictionary = magickMetadata.ClassPropertiesToDictionary();</example>

/// <returns>a dictionary of field names and values</returns>

public Dictionary<string, object> ClassPropertiesToDictionary()

{

return this.GetType()

.GetProperties(BindingFlags.Instance | BindingFlags.Public)

.ToDictionary(prop => prop.Name, prop => prop.GetValue(this, null));

}

#region AutoImplementedProperties

/// <summary>

/// used to show copyright notice NOTE the copyright symbol © gets automatically prepended

/// </summary>

public string Copyright { get; set; }

/// <summary>

/// use to show the name of the person who took the photo and/or user who uploaded

/// </summary>

public string Creator { get; set; }

public string Country { get; set; }

public string Headline { get; set; }

/// <summary>

/// Keywords NOTE if you call multiple times values will be appended not overwritten - call KeywordsClear() to reset

/// </summary>

public string Keywords

{

get

{

return \_keywords;

}

set

{

if (String.IsNullOrEmpty(\_keywords))

{

\_keywords = value;

}

else

{

\_keywords = \_keywords + “ “ + value;

}

}

}

public string Software { get; set; }

public string Source { get; internal set; }

public string Subject { get; set; }

public string Title { get; set; }

#endregion

// DISPOSE

public void Dispose()

{

\_image.Dispose();

}

}

}