INFO: Whitepaper On Adding Bates Number Stamp to Multi-page TIFF

A Surprisingly large number of moving parts

Use-Case

You have Multi-page TIFF files that need to have "bates numbering" applied to them

The good news is, Atalasoft can help.

However, you are going to need to make several key decisions about how you want to proceed - there are a few different standard ways to add such numbering to your images, and the way you go about it depends on your needs.

High Level Discussion

On the highest level, documents receiving a Bates number ("bates stamp") are often legal or other important documents and thus there are sometimes requirements not to alter the original content.

Why does this matter? because there are some technical issues with images.

Are you just going to draw the bates number directly over some part of the pages?

If so what is the chance that your number accidentally goes over some key bit of evidence/information?

Also even if that's OK, what happens if some unimportant information is where you put your bates stamp?? do you need to consider ensuring you put a properly sized blank background down first to block out the area where the stamp text will be?

OR do you need to alter the page to add a margin where the bates number is sure not to interfere?

Ok lets say you want a margin.. so there are two main ways you can do that

1. Resize the page to just add needed margin (thus increasing the size (height usually) of the page by some amount)
2. resample the page content to make it smaller within the original bounds so that the

original page size is preserved, but introducing "resample error:"

These are important questions with possible implications

"just slapping the number over some part of the page" could inadvertently cover important content unless you KNOW the pages have a given margin, using option 1 will ensure your original image is unaltered quality wise but the page is now some number of pixels / inches taller - if your pages were 8.5x11" and you added 1/3" of footer, your pages are now 8.5x11.33333"

Using the option 2 with resampling will of necessity resample the original image to be smaller - resampling is a highly technical subject but the key point is that there is no such thing as a "partial pixel" in a TIFF image.. so if you have a 2550px wide by 3300px image and you resample it so you have 100 pixels less of height, mathematically, the new page needs to be 2472.7272 x 3200 "Ok so you round and make it 2473 x3200 and call it a day" - sure that's easy and likely what will happen.. but .. here's the thing - that's just the size .. each pixel in the image needs to be evaluated and decisions need to be made about how to make new pixels based on where the new size vs old change things.. and this means many many such resize /scaling decisions each of which MUST result in an integer value and thus you get potentially 2 rounding decisions for each pixel in the image (2550 x 3300 = 8,415,000) so, 8 million pixels possibly need to be rounded in terms of their x and Y - 16 million calculations each with some amount of rounding...

We have multiple sophisticated resample algorithms in DotImage and I'm making it sound way more painful than it is in practice, but you must understand that there are absolutely unavoidable reasons that are fundamental to image processing on computers that come into play and using the ResampleCommand will introduce alterations to the content

Again, for some needs this is acceptable, but for some legal purposes, altering the images/document content could be considered inappropriate or even go against court practices and procedures. (This article is written by a support engineer, not a lawyer, ALWAYS discuss any such issues with an attorney)

Plan of Attack

So, before you get to the "doing" you need to really decide which approach you want to take

STAMP-OVER-ORIGINAL

INFO: Whitepaper On Adding Bates Number Stamp to Multi-page TIFF

You will jut write your bates number to an area of the document, either with or without a "frame" that can ensure the number is visible even if over content but at the cost of content

pros:

- Original image is not resampled/re-scaled
- Code-wise this is easiest

cons:

- potential for number covering content (framed number)
- potential for number being unreadable due to clash with content (unframed number)
- Technically this "modifies the original content" insofar as the number is added on top of some area of the original image content.

ADD MARGIN VIA RESIZE PAGE

Here, you add a margin (usually on the bottom) by resizing the page to simply add some amount of space needed. This will let you add known empty space where the number can be added and won't alter the original document content otherwise. However the page size is altered.

pros:

- No alteration / degradation / resampling of orignal content
- Programmatically quite straightforward

cons:

- Page size has to change - most commonly, you add 1/3 to 1/2 of an inch to the height
- page size changing can lead to weird issues printing or converting to other document formats

RE-SCALE ORIGINAL IMAGE SMALLER SO IT MAKES ROOM

By far, the most complicated approach. You essentially take the size of the original page, then resample the original page image small enough so that when placed on top of (overlaid) a new blank page the size of the original, it leaves you a margin, Then you bates stamp that margin.

This is most technically complicated because the statement about "resample the original image page small enough" is doing some serious heavy lifting..

ResampleCommand is incredibly powerful - we have several very good algorithms, but ultimately in imaging, there's no such thing as a "partial pixel" and thus when an image is resampled to sizes that are not conveniently perfect candidates introduce "rounding errors" as well as resampling error. Its not quite as bad as it seems here but you need to be aware that images (especially bitonal ones) are altered during resample and this can lead to unwanted loss of quality

pros:

- Final image sizes do not change

cons:

- Lots of moving parts
- Resampling errors introduced into image
- original content size shrinks to stay within original page size

IMPLEMENT THE STAMP

Regardless of what approach from above you use, every implementation of this requires that you be able to write the text of the bates number to your image... whether yu decide to give it a background frame or not (though you will see this is all still done in the same part of the code).

So, we're going to take as read that you have an AtalaImage object upon which you are ready to draw your bates number / stamp. You have either just given us the raw original page, or you've extended the bottom with a simple resize canvas or you've done the resample and overlay.. What matters here is we need to know

- The number to write
- The font you want to use
    - face
    - size
    - style
- whether you want a frame or not
- The location on the image to anchor your stamp

For this exercise we will assume black text over white

Note that this leaves a LOT up to you still to decide. You may need to adjust the font size and location quite a bit until you find a perfect combo that works for your methods - perhaps you want to scale your font so that you take the page size/resolution into account, perhaps you don't.

That is going to be beyond the scope of this discussion because we're already really going to be quite lengthy here. Perhaps we can revisit and extend at a later date.

So, lets apply a bit of an abstract here

Method signature

```
private void ApplyBatesStamp( AtalaImage targetImage, int stampNumber, Point location, System.Drawing.Font stampFont, bool addBackgroundBox) { }
```

Actually handling the stamp

So, there's still a bit of an "if bomb" here - if we do a background box, we need to calculate the size of the rendered text first and use that calculation to DRAW the frame then we DRAW the text on top.

To get the text part (given the above signature) and putting it together:

```
private static void ApplyBatesStamp(AtalaImage targetImage, int stampNumber, Point
location, System.Drawing.Font stampFont, bool addBackgroundBox) {

    Console.WriteLine("ApplyBatesStamp()");     Console.WriteLine("  targetImage: " +
targetImage.Size.ToString());     Console.WriteLine("  stampNumber: " +
stampNumber.ToString());     Console.WriteLine("  location: " + location.ToString());
Console.WriteLine("  stampFont: " + stampFont.ToString());     Console.WriteLine("
addBackgroundBox: " + addBackgroundBox.ToString());     Canvas canv = new Canvas(
targetImage );    string batesNumberString = stampNumber.ToString( "D6" );     Color
backgroundBox = Color.Transparent;     if (addBackgroundBox)     {         backgroundBox =
Color.White;     }     canv.DrawText(batesNumberString, location, stampFont, new
SolidFill(Color.FromArgb(128, Color.Red)), new SolidFill(backgroundBox)); }
```

The incoming targetImage is written ONTO which is why this does not return anything

That takes care of "how to use Atalasoft do write a bates number.

Are we there yet?

Well, not entirely - we have the "put pixels on image" but you probably noticed it requires you to provide it with a couple of pieces of info as mentioned.

We need to know how big the font should be and we need to know where on the page to put it.

We could use canvas.GetTextSize(batesNumberString, font) to get the size the box needs and then just position it say, "nice" placement might be starting about the box height from the left and giving the box just enough room to fix at the bottom

```
Point location = new Point( boxSize.Height, targetImage.Height - boxSize.Height);
```

but we'll just assume you know the location.

Scaling the font size can be a bit of a pain.. simple scaling indicates that a size of 36 is pretty good for an image at 200 DPI

- 54 is better for a 300 DPI image
- 27 for 150 DPI

- 24 for 100 DPI

Roughly... you'll need to scale - I came up with a quick scaling based on standard widths

Pass in image width and initial size that works for you at 200 DPI

I was calling it with 36.. that seemed to work

```
private static int SimpleFontScaling(int initialFontSize, int width)
{     Console.WriteLine("SimpleFontScaling()");     Console.WriteLine("  initialFontSize: " +
initialFontSize.ToString());     Console.WriteLine("  width: " + width.ToString());     double
scaleFactor = 1;     if (width > 3000)     {         // greater than 300 DPI         scaleFactor =
2;     }     else if (width > 2400)     {         // approx 300 DPI         scaleFactor = 1.5;     }
    else if (width > 1600)     {         // approx 200 DPI         scaleFactor = 1;     }     else if
(width > 1200)     {         // approx 150 DPI         scaleFactor = 0.75;     }     else     {
//really low res         scaleFactor = 0.5;     }     return (int)(initialFontSize * scaleFactor); }
```

Going Multi-page

Thus far this is just one image at a time... so how does one parse a full multi-page document.

We will assume you start with a staring number which may not be 0 or 1.. we will assume the first page is given that number and each gets that +1.

The TiffEncoder has a marvelous property called Append - so we can just loop over and write to the same image over and over.. the first time Append false and the rest true and it will politely just add as many pages up to the hard 4 GiB limit for the TIFF file spec (2 GiB in 32 bit)

We'll use a FileSystemImageSource to loop

Make careful note I do not call .Dispose() on the images but instead use fsis.Release(img) this is CRUCIAL for good memory management.

```
{     Console.WriteLine();
Console.WriteLine("=====================================================
    Console.WriteLine("StampMultipageTiff()");     Console.WriteLine("  inFile: " + inFile);
Console.WriteLine("  outFile: " + outFile);     Console.WriteLine("  batesStartNumber: " +
batesStartNumber.ToString());     TiffEncoder enc = new TiffEncoder();     enc.Append =
false;     using (FileSystemImageSource fsis = new FileSystemImageSource(inFile, true))
{         while (fsis.HasMoreImages())         {             AtalaImage img = fsis.AcquireNext();
            Console.WriteLine();
Console.WriteLine("--------------------------------------------");
Console.WriteLine("AtalaImage Acquired.. FrameIndex:" + (fsis.Current -
1).ToString("D3"));             Console.WriteLine("  Size: " + img.Size.ToString());
Console.WriteLine("  Resolution: " + img.Resolution.ToString());
Console.WriteLine("  PixelFormat: " + img.PixelFormat.ToString());             int fontSize =
SimpleFontScaling(36, img.Width);             Console.WriteLine("  fontSize: " +
fontSize.ToString());             Console.WriteLine("  BATES: " +
batesStartNumber.ToString("D6"));             ApplyBatesStamp(img, batesStartNumber, new
Point(20, img.Height - 100), new Font("Courier New", fontSize), true);
img.Save(outFile, enc, null);             fsis.Release(img);             enc.Append = true;
batesStartNumber++;         }     } }
```

We Have Finished with the Easy Bit

If you are OK with just stamping over then you may need to adjust size or placement some but this should "just work".

However, if you are under one of the other use cases, things are about to get a bit more complicated.. buckle in.

IMPLEMENT RESIZE PAGE TO ADD MARGIN APPROACH

OK so your use case allows you to modify the size of the pages themselves. This won't take too much more work - in fact you'll still use the original stamp code just the position will modify a bit and we need to do more

Single image concept and code

So, we need to make the image say half an inch taller.

So right away we run into the first "interesting problem"

what is "half an inch"?

That sounds like a dumb question, we all know what a half inch is (about 1.27 cm). but seriously, when I have an image that has some width and height in pixels, I need another piece of information: **Resolution** to know how that applies to inches or centimeters.

A full discussion of properly determining resolutions is best left for another article. so lets just over simplify it, shall we?

Common resolutions are 96 DPI (this is also a default of DotImage if DPI is not known), 100 DPI, 150 DPI, 200 DPI, 300 DPI. Most commonly, we run into 200 or 300 DPI images. Now in a "perfect world" if your document is US letter in Portrait orientation, then 200 DPI is 1700 x 2200 pixels where 300 DPI is 2550 x3300 pixels.

However, that assumes portrait (width < height) and it does not account for the very real world issues where scanners and faxes do not always give you images of pages that exactly fit - there's a bit of "fuzz" in the sizes, so rather than a long and complicated algorithm, lets simplify a bunch and we'll go with this:

```
static int CalcMarginHeight(int pageHeightInPx, double desiredMarginHeightInches) {
Console.WriteLine("CalcMarginHeight()");     Console.WriteLine("  pageHeightInPx: " +
pageHeightInPx.ToString());     Console.WriteLine("  desiredMarginHeightInches: " +
desiredMarginHeightInches.ToString());

    int marginHeight = (int)(desiredMarginHeightInches * 200);

    if (pageHeightInPx > 4000)     {         // assume 400+ dpi         marginHeight =
(int)(desiredMarginHeightInches * 400);     }

    else if (pageHeightInPx > 3000)     {         // assume 300 dpi         marginHeight =
(int)(desiredMarginHeightInches * 300);     }     else if (pageHeightInPx > 2000)     {
// assume 200 dpi         marginHeight = marginHeight = (int)(desiredMarginHeightInches *
200);     }     else     {         // Assume its like 96 DPI         marginHeight = marginHeight =
(int)(desiredMarginHeightInches * 96);     }     return marginHeight; }
```

So, now the "magic sauce" you need is to use the ResizeCanvasCommand.

This is the Atalasoft ImageCommand that change the size of the image while not disturbing the original content (it resizes around the image) so you have to provide an anchor point and a background color for the new canvas. It is used to make an image bigger and leave a margin.

We will calculate the new height by adding the MarginHeight to the original height and just add it to the bottom

AtalaIamge targetImage = new AtalaImage(inFile, 0, null);

// ask for a half inch margin int marginheight = CalcMarginHeight(int targetImage.Height, 0.5) ; ResizeCanvasCommand resize = new ResizeCanvasCommand(new Size(targetImage.Width, targetImage.Height + marginHeight), new Point(0,0), Color.White); AtalaImage resizedImage = resize.Apply(targetImage).Image;

// NOTE we dispose of original as ResizeCanvas changes size // and that means new image objet and we need to manage memory responsibly targetImage.Dispose();

Now, resizedImage has that nice empty margin.

You can apply the same logic above but now position the bates number in that margin ... here's the code:

```
            // Notice here I add  an extra 20 from the Y on the location so that its not right in
the top left of the margin

            ApplyBatesStamp(resizedImage, batesStartNumber, new Point(20, img.Height + 20
), new Font("Courier New", fontSize), true);

            resizedImage.Save(outFile, enc, null);

            resizedImage.Dispose();

            enc.Append = true;

            batesStartNumber++;

        }

    }

}
```

IMPLEMENT RE-SCALE ORIGINAL IMAGE SMALLER SO IT MAKES ROOM

We can't stress enough: this is the least recommended option

What if you have a situation where you are not able to change the page size - maybe all your pages are a very specific predefined size, or your printers are really weird about odd sizes etc.

There is another way forward, but it's by far the messiest and least pleasant. This is because it requires you to resample the original image down, potentially reducing quality due to resampling artifacts.

That's not to say it's super hard since we have already done all the hard work.. what we need to do here is simply take the final output of each page we did in the last section and resample it so it fits within the dimensions of the original.

It's easy in code because all we're going to do is create a new blank page the size of the original, resample the bates page with margin so that its height shrinks to fit the original size the width shrinks too), and finally place that via OverlayCommand onto the image.

INFO: Whitepaper On Adding Bates Number Stamp to Multi-page TIFF

This isn't that hard but it's really not a good idea - that ResampleCommand - we have several good resample methods but none of them will ever make the same quality image as the original unless you are resampling up to exactly 2x, 4x, 86, 16x etc original size. Down sampling to 1/2 , 1/4, 1/8 are the next best scenarios but it still means that a given group of 4, 8 16 etc pixels are compressed down to the next base 2 value - and given we're dealing with fairly small fractions here - this is not advised - use at your own risk

Concept and Code

To do this we need to know the original target pageSize in pixels, and we need to modified image with the margin.

```
private static AtalaImage ResampleAndPlaceResizedImage(Size origPageSize, AtalaImage
imageToPlace, bool disposeOriginalWhenDone = false)

{

    Console.WriteLine("ResampleAndPlaceResizedImage()");

    Console.WriteLine("  origPageSize: " + origPageSize.ToString());

    Console.WriteLine("  imageToPlace: " + imageToPlace.Size.ToString());

    Console.WriteLine("  disposeOriginalWhenDone: " +
disposeOriginalWhenDone.ToString());

    AtalaImage finalImage = new AtalaImage(origPageSize.Width, origPageSize.Height,
imageToPlace.PixelFormat, Color.White);

    ResampleCommand resample = new ResampleCommand(origPageSize.Height);

    AtalaImage overlayImage = resample.Apply(imageToPlace).Image;

    if (disposeOriginalWhenDone)

    {        imageToPlace.Dispose();     }     OverlayCommand overlay = new
OverlayCommand(overlayImage, new Point((int)( (origPageSize.Width -
overlayImage.Width) / 2), 0));     finalImage = overlay.Apply(finalImage).Image;
overlayImage.Dispose(); // done with this get rid of it before returning     return finalImage;
}
```

Now, lets add that code in to the previous.. we cut it in after the canvas resize margin and stamp applied but before we save it out:

```
            Console.WriteLine("fontSize: " + fontSize.ToString());

            Console.WriteLine("BATES: " + batesStartNumber.ToString("D6"));

            // so far we're same as before this is where the changes happen

            Console.WriteLine("Margin Height (inches): " + marginHeightInches.ToString());

            int marginHeight = CalcMarginHeight(img.Height, marginHeightInches);

            ResizeCanvasCommand resize = new ResizeCanvasCommand(new Size(img.Width,
img.Height + marginHeight), new Point(0, 0), Color.White);

            AtalaImage resizedImage = resize.Apply(img).Image;

            // NOTE that at this point we are done with the image source image so lets release
it now

            fsis.Release(img);

            // Notice here I add  an extra 20 from the Y on the location so that its not right in
the top left of the margin

            ApplyBatesStamp(resizedImage, batesStartNumber, new Point(20, img.Height +
20), new Font("Courier New", fontSize), true);

            AtalaImage finalImg = ResampleAndPlaceResizedImage(img.Size, resizedImage,
true);

            finalImg.Save(outFile, enc, null);

            enc.Append = true;

            batesStartNumber++;

        }

    }

}
```

Wrap-up

So, there you have it... 3 ways to apply a Bates Stamp to a multi-page tiff, each one building on the last and adding complexity.

This is by no means the only way to accomplish this task, it's merely a learning exercise by which you can hopefully see a way forward to building your own perfect for you bates numbering system. This is meant as a guide, not to hand you production code because each use case is different, each of the solutions provided here has pros and cons.

Hopefully this has given you insight into how to approach the problem.

Atalasoft Knowledge Base
https://www.atalasoft.com/kb2/KB/50443/INFO-Whitepaper-On-Adding-Bates-Numb...