HOWTO: Create a Custom BitonalFileSystemImageSource

Background

Customers often have a need to take an incoming single or multi framed file and convert all pages to true Bitona (Black and White only - no gray)

Converting a single image to bitonal is straightforward using one of the Threshold commands

- GlobalThresholdCommand
- DynamicThresholdCommand
- AdaptiveThresholdCommand

The pattern of use for these is roughly:

```
talaImage srcImg = new AtalaImage("someFile", frameIndex, null); // note, you can use any of the three commands from above here GlobalThresholdCommand threshold = new GlobalThresholdCommand(); AtalaImage bitonalImg = threshold.Apply(srcImg).Image; srcImg.Dispose(); // no longer needed // excample saving this as a one page tiff // if you want to save as other formats, change up the encoder and file extension bitonalImg.Save("outFile.tif", new TiffEncodeer(), null); bitonalImg.Dispose(); // cleaning up memory.. no longer needed
```

Now, instead of saving and disposing, bitonalImg will be a proper 1 bit per pixel image with "min is white" palate suitable for whatever additional processing you want

NOTE that the threshold commands all return a totally new object so the original image should be disposed

The only issue here is this is great for a single image / single frame but it is quite common to want to apply to all pages in a given file.

To deal with multiple pages, the use of an ImageSource is recommended

Suggested Approach for Multi Page Files

using a FileSystemImageSource is recommended for multiple page files.. this would be the normal approach(withouth converting to black and white)

```
tring inFile = @"C:\pathTo\input.tif"; string outFIle = @"C:\pathTo\output.tif"; using (FileStream outStream = new FileStream(outFile, FileMode.Create)) { using (FileSystemImageSource fsis = new FileSystemImageSource(inFile, true)) { TiffEncoder enc = new TiffEncoder(); enc.Save(outStream, fsis, null); } }
```

This works great (though nonsensical as we are "converting" the tiff into a tiff with no editing.. so how would we go about altering this to work with converting images?

```
tring inFile = @"C:\pathTo\input.tif"; string outFIle = @"C:\pathTo\output.tif"; // note, you can use any of the three commands from above here GlobalThresholdCommand threshold = new GlobalThresholdCommand(); using (FileStream outStream = new FileStream(outFile, FileMode.Create)) { using (FileSystemImageSource fsis = new FileSystemImageSource(inFile, true)) { TiffEncoder enc = new TiffEncoder(); //// instead of just direct encodr saving we will use a loop //enc.Save(outStream, fsis, null); while(fsis.HasMoreImages()) { outStream.Seek(0, SeekOrigin.Begin); // IMPORTANT! AtalaImage currentPage = fsis.AcquireNext(); using (AtalaImage convertedPage = threshold.Apply(currentPage).Image) { enc.Save(outStream, convertedPage, null); } fsis.Release(currentPage) enc.Append = true; // IMPORTANT } } }
```

what this has done is to frame by frame convert to bitonal and then use the TiffEncoder to append pages..

This works GREAT if you use TiffEncoder, but PdfEncoder does not have the ability to append. So how do we get this same idea but apply it to saving PDF?

The answer is we do the conversion inside a custom derived image source. In this case we will create a BitonalFileSystemImageSource by inheriting FileSystemImageSource and simply override the LowLevelAcquire to let us do arbitrary conversion.

BitonalFileSystemImageSource class

```
sing System; using System.Collections.Generic; using System.Text; using Atalasoft.Imaging; using System.Runtime.Serialization; using Atalasoft.Imaging.ImageProcessing.Document; namespace AtalasoftSupportUtils { public class BitonalFileSystemImageSource : FileSystemImageSource { public BitonalFileSystemImageSource(String path, Boolean doAllFrames) : base(path, doAllFrames) { } public BitonalFileSystemImageSource(String path, String pattern, Boolean doAllFrames) : base(path, pattern, doAllFrames) { } public BitonalFileSystemImageSource(String[] fileNames, Boolean doAllFrames) : base(fileNames, doAllFrames) { } public BitonalFileSystemImageSource(SerializationInfo info, StreamingContext context) : base(info, context) { } // this method overrides the LowLevelAcquireMethod in FileSystemImageSource protected override ImageSourceNode LowLevelAcquire(int index) { ImageSourceNode node = base.LowLevelAcquire(index); if (node == null) return null; AtalaImage image = null; // Force the image to B&W if it's not already if (node.Image.PixelFormat !=
```

```
PixelFormat.Pixel1bppIndexed) { // you can use any of our threshold commands here instead
instead if you wish ... example // GlobalThresholdCommand cmd = new GlobalThresholdCommand();
DynamicThresholdCommand cmd = new DynamicThresholdCommand(); image =
cmd.Apply(node.Image).Image; } // couldn't reduce if (image == null) return node; // don't
need original image anymore node.Image.Dispose(); node.Dispose(); //disposes node.Reloader.
By some reason, this method doesn't dispose the node.image too. return new
ImageSourceNode(image, new FileReloader(image)); } } }
```

Putting it to work

so, we have the custom image source now.. we use it almost identically to the way we first used the FileSystemImageSource - it is pretty much "drop in and go"

Note we are changing to sing a PdfEncoder this time though you could use a TiffEncoder if you wanted a tiff - that is the magic of the custom image source it doesn't care and will work for any encoder that supports multiple pages.

```
tring inFile = @"C:\pathTo\input.tif"; string outFIle = @"C:\pathTo\output.pdf"; using
(FileStream outStream = new FileStream(outFile, FileMode.Create)) { using
(AtalasoftSupportUtils.BitonalFileSystemImageSource bfsis = new
AtalasoftSupportUtils.BitonalFileSystemImageSource(inFile, true)) { PdfEncoder enc = new
PdfEncoder(); enc.SizeMode = PdfPageSizeMode.FitToPage; // IMPORTANT FOR PDF
enc.Save(outStream, bfsis, null); } }
```

Final Words

BitonalFileSystemImageSource examples for both PDF and VB.NET are attached to this KB article.