HOWTO: Determine an Image File Type / Format with DotImage

In the windows world, file extensions are a common means of identifying file content type

If you see a file named foo.txt, you would usually be safe in assuming that the file contains plain ASCII text

Likewise, if you have a file that is foo.png, you would usually be safe in assuming the file is a Portable Network Graphics (PNG) image file.

However, the reality is that the extension in Windows does tell windows "what program should try and open this" there's nothing to stop someone from renaming the file foo.png to foo.txt and thus causing the image file to be opened in Notepad (which is an ugly thing)

It should be noted that Atalasoft support has even run into customer cases where a customer has taken a file (foo.png for instance) and renamed its extension (for example renaming foo.png to foo.pdf) and then wondering why the file won't open in acrobat after they "converted to PDF".

So, obviously changing the extension of a file does not actually change the content type of the file.. it merely changes a part of the file name that Windows and other applications use to give a clue about what type of content the file has / which program to use to open the file.

Now, on one level, DotImage somewhat hides this from you... if you took the aforementioned foo.png file and renamed it say, to foo.jpg and opened it in one of our viewers or even just an AtalaImage:

```
AtalaImage img = new AtalaImage("foo.jpg");
```

It would open the file just fine (assuming the file was in the correct path and the image was actually a valid (png) image)

Basically, we ignore the file extension.

once we have the file as an AtalaImage object, we no longer care what type it was on disk.. you can effectively save it out to any supported type...

so

HOWTO: Determine an Image File Type / Format with DotImage

```
img.Save("out.png", new PngEncoder(), null);
```

would save it as a proper png image

```
img.Save("out.bmp", new BmpEncoder(), null);
```

would save it as a bitmap, and so-on

So, how dos DotImage determine what type of image a file is?

Welcome to the RegisteredDecoders class and its Decoders collection

When you use DotImage to read an image in one of our viewers or directly use classes such as FileSystemImageSource or AtalaImage, you pass it a filename or a stream containing the data you want to open.. and what DotImage does is that it iterates through every decoder it finds in

RegisteredDecoders.Decoders

and checks to see if the image is a type the current decoder knows how to handle... Once it finds a decoder that indicates it knows how to handle the data, it uses the decoder to render the image.

The way any decoder knows how to handle an image type is that the specific decoder knows how to read its supported type's header / data information / structure to say "ahh this is a type I know how to handle"

Internally, in a given image file, toward the beginning of the file there will be a byte sequence that identifies the content.. for a TIFF file that data looks like

II* (in hex, it would be bytes 0x49 0x49 0x2A at the very beginning of the file)

For PDF it might be

%PDF-1.6 (in hex, it would be bytes 0x25 0x50 0x44 0x46 0x2D 0x31 0x2E 0x36 or similar at the beginning of the file)

So, we're actually reading the image data header from the file and determining what decoder we have that knows it.

HOWTO: Determine an Image File Type / Format with DotImage

This becomes relevant because it answers the question "when I open this file in DotImage, why do I get a 'Unrecognized file type' error?"

Unrecognized Image Type Exception

The answer is "because the file you opened did not contain data that any ImageDecoder in the RegisteredDecoders.Decoders collection knew how to handle"

Now, this could be for a number of reasons

1) if you were opening a stream object but the stream was not at the 0 / start position.. to fix this, always ensure you reset the stream before trying to read it

```
myStream.Seek(0, SeekOrigin.Begin);

AtalaImage img = new AtalaImage(myStream);
```

2) Something in your code has removed the needed decoder (you should take care never to remove any decoders from RegisteredDecoders.Decoders and should NEVER call

```
RegisteredDecoders.Decoders.Clear();
```

3) The decoder your image needs is one that we do not include in the base implementation either because it's uncommon or because it's an add-on which requires additional licensing such as PdfDecoder, DwgDecoder, DicomDecoder, Jb2Decoder, etc..

In order to add support for a given extra image type you must ensure you have added its decoder to the RegisteredDecoders.Decoders collection. You should do this in a static constructor for your class/app so that it is done once, and only once to avoid double-adding

```
static Form1()

{

    RegisteredDecoders.Decoders.Add(new PdfDecoder() { Resolution = 200 });

}
```

in VB this is done in a Shared Sub New

```
Shared Sub New

     RegisteredDecoders.Decoders.Add(new PdfDecoder() With { .Resolution = 200
})

End Sub
```

4) Your file is corrupt/damaged or is simply not of a type that we support

Determine Image Type Programmatically

There is a practical question that comes up quite often: "I have need to programmatically know what type of image the file was.. maybe because I want to apply special processing that is only valid for certain types.. such as using a PdfAnnotationDataImporter (which will error out if I try and use it on a file that is not a valid PDF)"

The RegisteredDecoders class again comes to your rescue... this time with the

RegisteredDecoders.GetDecoder(...) method.

NOTE: do NOT be tempted to use the ImageType property of ImageInfo as the ImageType is not aware of any additional decoders.. this is deprecated.... ImageType typ = RegisteredDecoders.GetImageInfo(...).ImageType;

Don't use the ImageType enumeration at all... please.

You can get the decoder that the collection has found for your file and use reflection to get the type and decide what actual type the image is.. here's a simple example just to say "is this a PDF?

```
        if (rawDec !=null && rawDec.GetType() == typeof(PdfDecoder))
        {
            return true;
        }
    }
    return false;
}
```

Keep in mind that some decoders such as RawDecoder and OfficeDecoder support many different file types.. OfficeDecoder supports MS word, Excel, and PowerPoint 97 and later as well as RTF, and the RawDecoder supports raw formats from many different vendors ... all of which are classified as "raw image formats" but are technically entirely different from each other internally

So, with DotImage if you give it a word or excel file, it will come back with OfficeDecoder, but you won't know which type of file it was (and you will likely need to fall back on the file's original extension to give you the hint you need)

Original Article:

Q10445 - HOWTO: Determine an Image File Type / Format with DotImage

Atalasoft Knowledge Base
https://www.atalasoft.com/kb2/KB/50056/HOWTO-Determine-an-Image-File-Type-F...