Atalasoft's Five-Day Imaging e-Course

This is part one of five of Atalasoft's introduction to Enterprise Imaging. This five day course will cover the following topics:

1. **Understanding Image Formats**
2. Enterprise Imaging on the Web
3. Scanning from the Web
4. Extracting Information from Images
5. Performance Tuning

Chapter 1: Understanding Image Formats

We're all somewhat familiar with the common image formats such as JPEG, PNG, and GIF, but there are over a hundred image formats. Many are legacy, but there are still reasons to choose among a dozen or so of the most common ones, depending on your application.

There are trade-offs between the various formats and understanding them will allow you to optimize your storage requirements while still maintaining the highest quality images possible. When trying to figure out which format to pick for a given use, ask the following questions:

1. How are the images **originally captured** from the imaging device (e.g. scanner, digital camera, medical device)?
2. Are the images in color? If so, **is the color important** to maintain?
3. Is the image photographic content with **rich color**, graphic art with **few colors**, or a **black & white text** document?
4. What is the DPI and the **number of pixels** in the image?
5. Do I need a format that can hold **multiple images** (a multi-page or multi-frame format), such as for a scanned document that I want to keep in a single file?
6. What **deployed software** must be able to view the image?
7. Will the image be only viewed on screen or **will it be printed**? If printed, at what DPI?
8. How important is it to maintain the **original quality**?
9. How much **storage are you allocating** per image?
10. What is the **budget for purchasing toolkits** if the format isn't supported by your language's standard library?

Image Format Terms

Before we get into the attributes of the different formats, here are some of the basic terms that are used to describe image formats.

Codec: *Codec* stands for Coder-Decoder, and it is packaged code that can read and write a specific format. The word

codec is used for more than image formats (e.g. video and audio also use the term). Usually it's used in conjunction with the format name, such as "JPEG Codec" or "TIFF Codec". Creating the file is called *encoding* and reading the file is called *decoding*.

Resolution: The *resolution* of an image determines the size of the image when it is displayed or printed at actual size. Often this is expressed in Dots Per Inch (or DPI), but some formats can support metric or other expressions of resolution. The vertical and horizontal resolutions are often the same, but they don't need to be. For example, if the device doesn't render or capture square pixels, like many fax machines, then the two resolutions will be different.

Channel: A *channel* is a component of a color. To make the final color, the channels must be combined. Common channel standards are RGB (Red/Green/Blue) and CMYK (Cyan/Magenta/Yellow/Black). There is a special optional channel called the **Alpha channel**, which, if present, determines the transparency of the color.

Color Depth: The *color depth* describes how many bits are used to represent a pixel's color, expressed in bits per pixel. It is a component of the pixel format.

Frame: The term *frame* is sometimes used to refer to an individual page in a format that can hold multiple pages.

Palette: A *palette* is a list of colors that are used in an image. A palette may be used to save space. For example, it typically takes three bytes to store a color (one for each the red, green and blue channels). But if an image has only 256 different colors, it's better to use just one byte to index into a list of those colors. Doing so would mean that the image could be represented with approximately one-third of the required storage. When you represent an image this way, it may be referred to as a "paletted" or "indexed" format.

Pixel Format: A *pixel format* describes how each individual pixel of an image is represented. The standard way is to describe how many bits are used and what those bits represent. For example, the pixel format "24 Bits per pixel (BPP) BGR" means that 24 bits are used to represent a pixel and that 8 bits are used for each the "Blue", "Green", and "Red" channel, and that they are stored in that order. The format "1 BPP Black and White" means that 1 bit is used and that it represents either black or white. Other formats include "8 BPP Gray" (one 8 bit channel of gray), "8 BPP Indexed" (on 8 bit index into a palette of 256 colors), "32 BPP BGRA" (four 16 bit channels of each Blue, Green, Red, and Alpha).

Metadata: *Metadata* is all of the extra information about an image that is not the image's pixel data. There is always a base level of information like the height, width, pixel format, or resolution, but there might also be information about the camera, geographic coordinates from a GPS, original lighting conditions, and even a thumbnail of the image.

Lossy vs. Lossless: A *lossy format* is one that sacrifices image quality for a compression benefit. This means that some of the information is irretrievably lost when the image is saved into a lossy format, and each time the image is loaded and resaved, more information will be lost. Often, the amount of loss is controllable, some loss will happen on each save. A *lossless format* is one that uses a compression technique that is completely reversible, so that all of the quality in the original image is retained.

Common Formats

JPEG

The JPEG format was designed to be good at compressing photographs. To make the files small, JPEG uses a lossy compression method.

Basically, a JPEG encoder breaks up the image into 8x8 cells, and then each cell is converted into a function that approximates the data in the cell and assumes a uniform brightness. Storing the attributes of the function uses approximately 10% of the space of the actual pixel data, but there will be a noticeable difference if the brightness of the colors within the cell varies greatly. Since a standard document has the maximum variation of brightness (black and white), there are usually very noticeable distortions of the color around the text.

To control the encoding, you can provide a quality parameter that can be set from 0-100. Keep in mind that choosing a quality of 100 does not mean that you won't lose quality. If you repeatedly edit the JPEG and save it, the encoding process is used again and will result in more loss. If necessary, it is possible to make some kinds of edits by directly manipulating entire cells without decoding and re-encoding them – for example you could remove cells around the outside to result in a crop, or move and rotate individual cells to do a 90 degree rotation of the entire image.
JPEG is often used inside of other formats that allow a choice of compression methods.

Advantages of JPEG

1. Results in small files for photographic content
2. Almost any standard image viewer can view them, including web browsers
3. Automatically created by digital cameras
4. Can hold descriptive and custom metadata, which is often inserted by cameras
5. JPEGs can be created so that they can be progressively decoded – meaning that if you are transferring them over a slow connection, the receiver can read the beginning of the file and start showing it, and show more as more is downloaded.
6. Random portions of the JPEG can be decoded without decoding the entire image

Disadvantages of JPEG

1. Cannot accurately represent black and white documents
2. Cannot accurately represent images with greatly varied brightness, such as graphic art
3. Each edit and save will result in more loss of quality
4. The JPEG file can only hold a single image

PNG

The PNG format was designed to replace GIF. It is better in many ways (GIF only supports an 8-bit palette, has simple transparency support, and is usually bigger), but the main reason PNG was developed was because parts of GIF were patented and required a license fee (the patent has since run out).

PNG files can accurately represent any image, but will result in significantly bigger files than JPEG for photographic

content. PNG is a good interchangeable format while an image might still be edited or if retaining quality is extremely important. It is also good for use on the web for graphic images, especially those that use the alpha channel to blend the image in with the background.

Advantages of PNG

1. Results in small files for graphic content
2. Supports a full alpha channel
3. Retains all of the quality of the original image
4. Almost any standard image viewer can view them, including web browsers (You need to use a workaround for IE6 if your PNG has an alpha channel)

Disadvantages of PNG

1. Photographic images will be larger than JPEG
2. The PNG file can only hold a single image

TIFF

A TIFF file is a container of compressed images and metadata. Each individual frame of the TIFF can be compressed with one of a few different compression methods depending on the nature of the image in the frame. Since TIFF files can contain more than one image, they are a very popular way to store scanned documents and faxes.

The compression formats available include JPEG, LZW (similar to GIF), Flate (similar to PNG), and a few that are tuned for 1 bpp images (Group 3, Group 4, and a Huffman based RLE). There is also the option for a simple compression or no compression if decoding and encoding speed is important. In addition, individual frames can be stored as strips or tiles so that parts of an image can be accessed without requiring the entire frame to be decoded.

Because of ambiguity in the original specification, there was a proliferation of dialects, especially with concerns to what is called "Old-Style" JPEG (replaced with "New-style"). The standard Codec that comes with Windows XP and Vista doesn't even try to read "Old-style" because of the problems in specifying what that exactly is. Support for the various old-style variants is usually only found in third-party toolkits and it is strongly encouraged that no one create more old-style TIFF files.

Advantages of TIFF

1. Can contain multiple pages
2. Each page can be compressed based on its content
3. If the TIFF is stored in strip or tiled format, random portions can be accessed (resulting in fast display if the viewer takes advantage of this)

Disadvantages of TIFF

1. Some parts of the specification were ambiguous and led to proliferation of bad TIFF creators and consequently bad TIFF's. Many standard codecs cannot read all of the variations.
2. Usually requires special software to view, and many machines do not have a TIFF viewer pre-installed.

3. Web browsers cannot show a TIFF directly

PDF

PDF was originally developed by Adobe as an archival format for documents, but is often used to hold images because it can contain multiple images. PDF is a well-specified format, and the Acrobat Reader or equivalent is installed on nearly every machine and many mobile devices. Like TIFF, PDFs can contain multiple pages, and each individual page can be compressed based on its content.

Like TIFF, PDF can store an image as JPEG, Deflate (similar to PNG), Group 4. In addition, it supports two advanced codecs (described in detail below). The first, JPEG 2000, is for photographs and loses much less quality compared to JPEG for the same file size. The second, JBIG2, is tuned for 1-bit black and white images.

Since PDF is designed to contain more than just images, they are ideal for storing the results of images that have had optical character recognition (OCR) performed on them. The recognized text can accurately be placed behind the pixels representing the text. If this is done, the resulting PDF is called a Searchable PDF, and search engines will index them, and some viewers will allow you to select and search for text within them.

Advantages of PDF

1. Can contain multiple images
2. Can contain non-image content in addition to images (to make searchable PDFs for instance)
3. Viewers exist on nearly every machine, though web browsers cannot view them directly
4. Can use advanced codecs (JPEG 2000 and JBIG2)
5. Specifies a format for storing annotations in addition to the image

Disadvantages of PDF

1. Cannot be viewed directly by browsers
2. Can be larger than TIFF for the same images with the same compression
3. Editing, reading, or saving PDF is usually not part of a programming language standard library

Advanced or specialty formats

RAW

RAW is not really a standard, but instead, a collection of formats, loosely specified by individual digital camera manufacturers. It was created to give a lossless option for cameras to use so that you could manipulate images before they were converted to JPEG. The RAW format does not encode pixels, but instead encodes the original readings on the camera's sensor.

A version of the RAW format, called DNG, has been introduced in an attempt to standardize across all camera manufacturers.

Advantages of RAW

1. Useful to get them instead of JPEG if the photographs are going to be edited
2. Digital cameras can create them directly

Disadvantages of RAW

1. Specifications don't exist and camera manufacturers each have different formats
2. Viewers are not commonly installed, and web browsers cannot view them directly
3. Can only encode a single image
4. The files can be very large

JPEG 2000

JPEG 2000 is a lossy format similar to JPEG (although lossless versions are part of the standard). It uses more processor time to encode and decode, but the resulting files are higher quality for the same file size of JPEG. Unfortunately, viewers aren't commonly installed, so the best way to use JPEG2000 is inside of a PDF (which adds some overhead).

In order to benefit from the JPEG2000 format, the images should be relatively large or grayscale. Low resolution color images often compress just as well with JPEG format as JPEG2000.

Advantages of JPEG 2000 over JPEG

1. Lossy version retains more quality
2. Lossless versions are supported by viewers that support the lossy version

Disadvantages of JPEG 2000 compared to JPEG

1. Viewers aren't installed on most machines
2. Web Browsers cannot view them directly

JBIG2

JBIG2 is designed for multipage 1 bpp images, and comes in lossy and lossless variants. Its most common usage is inside of PDF, since viewers aren't widely installed. According to the creators, JBIG2 can result in files that are one-third the size (or smaller) of Group 4 (the best available standard 1-bit in TIFF and PDF). Unfortunately, to get most of the benefits of JBIG2, you need to be able to segment the original image into regions (such and text or image) and take advantage of highly tuned compression.

Advantages of JBIG2 over TIFF or PDF with Group 4

1. Can result in smaller files

Disadvantages of JBIG2 compared to TIFF or PDF with Group 4

1. Viewers are rarer than TIFF, and far less likely to be installed than a PDF viewer

DICOM

DICOM is the standard image format for medical devices. Similarly to RAW, it encodes the original data supplied by the machine, not the pixel data. It supports a variety of compression standards for gray and color data and typically uses more than 8 bits per channel.

DICOM images have multiple frames and each frame can use a different compression. There are lossy and lossless choices available. JPEG and JPEG 2000 are commonly used inside of DICOM.

If you are creating a medical device or need to interoperate with DICOM devices, you certainly should use this format. Otherwise, there is very little reason to.

Summary

Format	*Ubiquity of Viewers*	*Viewable in Web Browsers*	*Multipage*	*Supports 1-bit formats*	*Lossy v. Lossless*	*Best Imaging Use Case*
JPEG	High	Yes	No	No	Lossy	Photographs for the web
PNG	High	Yes	No	Yes	Lossless	Graphical elements for the web that use an alpha channel
TIFF	Somewhat	No	Yes	Yes	Either	Multipage documents that come from a scanner or fax
PDF	High	No	Yes	Yes	Either	Archival format for images with recognized text
RAW	Low	No	No	No	Lossless	Capture format for photographs that will be edited
JPEG 2000	Very Low	No	No	No	Either	Inside of PDF for photographs
JBIG2	Very Low	No	Yes	Yes	Either	Inside of PDF for black and white documents
DICOM	Very Low	No	Yes	No	Either	Medical Images

Sponsor

This e-course is sponsored by Atalasoft, which publishes a .NET Imaging SDK called DotImage. Using DotImage you can work with all of the above formats and many more. Provided in the toolkit are controls for displaying and manipulating documents and images in .NET Windows Forms, ASP.NET, WPF, and Silverlight. We even have a way of viewing all of these formats in standard browsers without plug-ins, ActiveX, Java or Flash.

Atalasoft's Five-Day Imaging e-Course

This is part two of five of Atalasoft's introduction to Enterprise Imaging. This five day course will cover the following topics:

1. Understanding Image Formats
2. **Enterprise Imaging on the Web**
3. Scanning from the Web
4. Extracting Information from Images
5. Performance Tuning

Chapter 2: Enterprise Imaging on the Web

In Chapter 1, we learned that the common document image formats (TIFF and PDF) aren't directly viewable on the web. By directly viewable, we mean that you cannot put an IMG tag onto your web page that refers to a TIFF or PDF. Only JPEG, PNG, and GIF are supported across the major browsers.

So, if you need to view any of these other formats in a browser application, you have a few options.

1. Have the user download the image and **view it in external software**
2. Create an **ActiveX control or plug-in** for the browser that can show the image
3. **Convert the images** to JPEG, PNG or GIF and store them that way
4. Assume that they have Acrobat Reader and **convert the images** to PDF and store them that way
5. Create a server process that automatically **converts to JPEG, PNG, or GIF on the fly**

Option 1: Use External Software

The easiest thing to do is simply to create an anchor tag that links directly to the image file. When the user clicks it, the browser will download the file and open it in whatever viewer the user has configured to view that kind of file.

Advantages

1. Easy to program.
2. Low server load.
3. Works on every browser.
4. Works on some mobile devices, depending on the image.

Disadvantages

1. Requires the user to have viewing software previously installed.
2. If they annotate or edit the image, there is no easy way to save.
3. Requires that the entire file is downloaded before it can be viewed.
4. Hard to control distribution of the original content.
5. Not easy to control checkin/checkout for collaboration.
6. No way to have simultaneous live collaboration.
7. Viewers aren't integrated into the site.

Option 2: ActiveX or Browser Plug-in

All browsers support some kind of plug-in system that allows any programmer to extend the capabilities of the browser. It's not common for most developers to have created them, especially web developers, and may require C or C++ to get all of the advantages. In

In addition, there are some plug-ins, such as Java, Silverlight and Flash, that might already be installed and can download and run your code. Java has built-in support for many more image formats, but not PDF, and unfortunately, it isn't installed on most desktops already. Flash has much more penetration, but cannot show TIFF or PDF. Silverlight doesn't have the penetration of Flash and currently supports only JPEG and PNG.

Advantages

1. Low server load.
2. Can integrate editing, annotating and saving.
3. If Java is already installed and acceptable, can be easy to integrate.

Disadvantages

1. Hard to deploy.
2. Plug-ins are neither cross-browser nor cross-platform.
3. Requires user acceptance and administrative privileges to install.
4. There are no standard plug-ins that have high penetration and can show more formats than the browser.
5. Requires that the entire file is downloaded before it can be viewed.
6. Not supported in all mobile browsers.
7. Not easy to integrate well with the rest of the site or updateable by web designers.

Option 3: Convert and store the images in a web-viewable format

Since browsers can view PNG and JPEG natively, some choose to convert their document images to these formats so that they can simply use IMG tags in their web application. This is extremely easy to program but neither PNG or JPEG is a good choice for document images. Neither format is multipage, nor can you store 1-bpp images in them efficiently.

Advantages

1. Low server load for viewing, the conversions will take time.
2. Easy to program.
3. Works in every browser.
4. Works on mobile browsers.
5. Also makes it possible to use Flash or Silverlight.
6. Can integrate editing, annotating, and saving.
7. Easy to integrate and manage with the rest of the web application.
8. It's possible to view pages without downloading the entire document.

Disadvantages

1. Need to have a scheme for tying pages together.
2. Increased storage cost because you can't use ideal formats for greater compression.
3. May not comply with company policy if you need to maintain original files.
4. For some advanced formats, you will lose a lot of quality unless you maintain the original (e.g. DICOM and RAW have device readings which are more accurate than the pixels in an image)
5. Converting PDF would lose all non-image information (like text).

Option 4: Convert and store the images in PDF

Since most users will have Acrobat Reader or some kind of PDF viewer installed, you can combine option 1 with option 3 by converting the image to a PDF that the user downloads and views. You may also create *linearized PDF*, which is a variant of PDF that can be read before it is fully downloaded. This makes it possible to view the first page of the document without waiting for the download to complete.

Advantages

1. Low server load for viewing, the conversions will take time.
2. Easy to program.
3. Works in every browser.
4. Works on mobile browsers with PDF viewers.
5. It's possible to view pages without downloading the entire document.

Disadvantages

1. Increased storage cost because you can't use ideal formats for greater compression.
2. May not comply with company policy if you need to maintain original files.
3. For some advanced formats, you will lose a lot of quality unless you maintain the original (e.g. DICOM and RAW have device readings which are more accurate than the pixels in an image)
4. The user would need a full installation of Acrobat to add annotations.
5. If they annotate or edit the image, there is no easy way to save.
6. Hard to control distribution of the original content.
7. Not easy to control checkin/checkout for collaboration.
8. No way to have simultaneous live collaboration.
9. Viewers aren't integrated into the site.
10. Creating PDF may require third-party SDK's.

Option 5: Convert to a web-viewable format on the fly

One nice way to get the advantages of converting to a browser supported format without losing the fidelity of your originals is to convert them to a web-viewable format on the fly. The easiest way to do this is to use a CGI script, web control, or HTTP handler (which ever makes the most sense for your application).

The control is given the location of the original and converts it to either a PNG or JPEG and serves it back to the browser. Since it already has control of the process it could also provide thumbnails of pages on demand or break the image into tiles to save bandwidth.

Using this option (converting on the fly) with the previous one (converting to PDF) is an excellent way to support printing on the web. You will have total control of the printed output, and you could even script Acrobat Reader to start the print job automatically.

Advantages

1. Works in every browser.
2. Works on mobile browsers.
3. Could also be integrated with Flash or Silverlight.
4. Can integrate editing, annotating, and saving.
5. Easy to integrate and manage with the rest of the web application.
6. It's possible to view pages without downloading the entire document.
7. You can thumbnail or tile images to save bandwidth.
8. When zooming out or thumbnailing 1-bpp pages, you could convert to gray for nicer looking rendering.
9. Can easily integrate AJAX to request thumbnails on demand.

Disadvantages

1. Requires complex programming or a third-party tool.
2. Requires smart caching to keep the server load low.

If you'd like to see a video or use a demo of a website that uses this method, click this link:

http://www.atalasoft.com/imaging-course/chapter-2-more/

Summary

Method	Ease of programming	Works in all machines	Maintains original file	Can view a page without full download	Saves bandwidth with thumbs and tiles
Download File	Easy	No	Yes	No	No
Plug-in	Difficult, but third-party controls are available	No	Yes	No	No
Pre-Convert to Web Format	Easy	Yes	No	Yes	No
Pre-Convert to PDF	Easy, but may require third-party SDK's	Almost all (need a PDF Viewer)	No	Possibly, with linearized PDF	No
Convert on the fly	Difficult, but third-party controls are available	Yes	Yes	Yes	Yes

It's also possible to combine these options for different purposes. For example, convert on the fly for previewing and live annotating, but convert to a PDF for printing.

Sponsor

This e-course is sponsored by Atalasoft, which publishes a .NET Imaging SDK called DotImage. Using DotImage you can view and annotate any image in standard browsers without plug-ins, ActiveX, Java or Flash. Our PDF generation support includes a way to script Acrobat to print with a single line of code.

Atalasoft's Five-Day Imaging e-Course

This is part three of five of Atalasoft's introduction to Enterprise Imaging. This five day course will cover the following topics:

1. Understanding Image Formats
2. Enterprise Imaging on the Web
3. **Scanning from the Web**
4. Extracting Information from Images
5. Performance Tuning

Chapter 3: Scanning from the Web

In Chapter 2, we learned about different ways to incorporate TIFF and PDF into a web application, specifically for viewing, editing, annotating and saving the documents. Another important aspect of Enterprise Imaging is capture, specifically from scanners, and this chapter will address the options you have when incorporating scanning into web applications.

Unfortunately, there is no standard for interacting with a scanner via a browser, and it's not supported in HTML or JavaScript. So, if you need to scan a document in a browser application, you have a few options.

1. Have the user **download and install** an external program.
2. Create an **ActiveX control or plug-in** for the browser that can scan.
3. **Scan from the server.**
4. Initiate scanning from a **network scanner**.

And, there are a few basic scenarios that you might need to support:

1. Scanning an entire **new document**
2. Scanning **multiple documents at once** with separator pages
3. **Re-scanning pages** to replace ones in a scanned document
4. Scanning in pages to **append to an existing document**

The solution that's right for you depends on which of these scenarios you have to support, where your scanners are installed, and how easy it is for you to push out installations to desktops.

Option 1: Create an external program for scanning

Any kind of client-side scanning is going to require something to be installed on the desktop machine, so making that explicit might be a good way to go. You could write an application that scans the documents and uploads them to your site. Then, create an installer that makes getting it on the desktop as easy as possible. Once the application is installed, scanning can be initiated at any time, even if the user isn't using the web application.

For .NET developers targeting Windows and IE, you can opt to deliver the application via click-once, which is a technology for easily deploying and updating WinForms based applications.

Advantages

1. Can be launched directly by the scanner.
2. Can be used even if the user isn't using the web application.
3. May be possible for IT to push out to desktops for internal deployments.

Disadvantages

1. Hard to integrate with the workflow of the web application, because it cannot be integrated into the web application's UI.
2. Needs to be ported to each platform.
3. Authentication with the website can be tricky from the application.
4. Cannot do page re-scanning and appending easily.

Option 2: ActiveX or Browser Plug-in

All browsers support some kind of plug-in system that allows any programmer to extend the capabilities of the browser. Unfortunately, none of the popular plug-ins (Java, Flash, Silverlight) support scanning, so you can't just use one of them.

Plug-ins can be challenging for web developers to create, but if you are targeting just IE, you can use a .NET WinForm Control, which is easy to write. Normally, it would not be a good choice to implement a part of your web-application with a plug-in, but it is acceptable here since there is no other way to have interactive scanning that is hosted in the browser.

Note: Atalasoft provides a very easy-to-use tool for scanning from a web portal. It's called WingScan and more information can be found at www.atalasoft.com/products/wingscan - ask us for a demo!

Advantages

1. The only way to integrate client-side scanning into the workflow of the web-application.
2. For intranets, the IT department can push out to all desktops and enable it.
3. The control is scriptable via JavaScript and the website can receive progress and other events via JavaScript as well.

Disadvantages

1. Plug-ins are neither cross-browser nor cross-platform.
2. Requires user acceptance and administrative privileges to install if IT isn't pushing it out.
3. There aren't any standard plug-ins that can scan.

Option 3: Scan from the Server

If you have network scanners that can be controlled remotely, then you can install the drivers on your web server and interact with the scanner from there. The major benefit is that you don't have to install anything on the individual desktops. Since the users still need to load paper into the scanners, they need to be physically accessible to them, so this is only useful if the server can interact over the network with scanners that are near the users.

Advantages

1. Works without installing anything on the desktop.

Disadvantages

1. Requires having a network scanner.
2. Cannot scan from a scanner connected to your desktop.
3. Scanner must be accessible via the network.
4. Some part of the server network needs to be physically accessible to the user, so they can get to the scanner.
5. You have to disable and re-implement any of the user-interface that the scanner driver needs to display.

Option 4: Initiate scanning from a network scanner

Similar to scanning from the server except that instead of initiating the scan from the server via a driver, you go over to the scanner and initiate the scan from there. Network scanners can put the scanned document somewhere on a file server, and then you have to have a server process that monitors that location and puts the file into your document database.

Advantages

1. Doesn't require any installation on the desktop.

Disadvantages

1. Requires network scanner capable of putting files on your file server.
2. Cannot scan from a scanner connected to your desktop.
3. Scan cannot be initiated from the web application.
4. Hard to authenticate with the website if necessary.
5. Cannot do page re-scanning and appending easily.

Summary

Method	Integrates with Web Application	Support page re-scanning and appending	Works without any installation	Can scan from desktop Scanner	Easily authenticates with web application
External App	No	No	No	Yes	No
Plug-in	Yes	Yes	No	Yes	Yes
Server-side scanning	Yes	Yes	Yes	No	Yes
Scan from Network Device	No	No	Yes	No	No

Sponsor

This e-course is sponsored by Atalasoft, which publishes a .NET Imaging SDK called DotImage and a TWAIN Web Scanning SDK called WingScan. Using WingScan, you could implement web scanning very easily. Ask us for a demo! More information can be found at www.atalasoft.com/products/wingscan

Atalasoft's Five-Day Imaging e-Course

This is part four of five of Atalasoft's introduction to Enterprise Imaging. This five day course will cover the following topics:

1. Understanding Image Formats
2. Enterprise Imaging on the Web
3. Scanning from the Web
4. **Extracting Information from Images**
5. Performance Tuning

Chapter 4: Extracting Information from Images

In Chapter 3, we saw the various ways in which we might scan documents from web-applications. Once a document is scanned we usually end up with a TIFF or image PDF files. Before we put them in the repository (or at least soon after), it's a good idea to try to extract as much information from them as possible so that we can:

1. Classify the document for **routing.**
2. Index the document for later **search retrieval.**
3. **Split the document** based on separator pages.
4. **Extract the fields** of a form to populate a database.
5. **Extract metadata** to help with search and viewing.

In order to do this, here are some of the things you should consider:

1. Before extracting data, **clean and pre-process the documents** to increase the accuracy
2. **Read barcodes** if present
3. **Extract metadata** and use this information to populate fields that will help find the document later
4. **OCR the document** and use the text for indexing or creating a searchable PDF
5. **OMR the document** and create records in your database for later tabulation
6. **Some processing must be done by a person**, aided with imaging software

Pre-processing Documents

Many information extraction algorithms (e.g. OCR) only work on 1 bpp documents. If you are scanning in color or grayscale, you have to convert to 1 bpp in a way that preserves as much of the information as possible. This is called *thresholding* or *binarization*. There are many ways of doing this, but the basic idea is that for each pixel in the image, you have to determine if it is foreground or background (i.e. black or white).

- The **simplest algorithm** uses a single brightness level for the entire document, darker pixels are foreground, lighter are background. It is the fastest algorithm, but it only works when you have very consistent pages and can know a good level without looking at the image and is useful only for black text on white pages.

- To improve on that, take a **global histogram** of the entire document and try to determine a brightness level that partitions the dark and light areas. Then use that level in the same way as the simple algorithm does. This algorithm is fast and works on pages with clear dark and light areas that don't differ in brightness across the page. It works well for black text on white pages.

- Most documents benefit from an **adaptive algorithm** that partitions the document by considering small areas and then using a histogram of just that area to determine a level for the pixels at the center of the area. This algorithm gives the best results over a large class of documents, and there are variants that can preserve large dark shapes that might otherwise have parts removed. It works well for black text on white pages, and does ok for photographs.

- If your document is a photo, then use **dithering**, which thresholds photographs well. It uses areas of dense black dots to represent dark colors and sparse black dots to represent light colors. Dithering is commonly used in black and white newspaper photos but is not good at thresholding text.

Before extracting information from a document, you should make sure the image is as clean as possible. Many scanners can do this automatically, and result in clean documents right from the start. If your scanners do not do this, almost all capture software has some cleaning capability. Here is a list of the types of cleaning algorithms you usually see:

- **Despeckle**: Used to remove very small stray dots (usually at most 3 contiguous pixels). These dots may be the result of dust on the capture device or an artifact of thresholding.

- **Deskew**: can detect and correct slight rotations in the document that occur if the paper wasn't loaded perfectly.

- **Border removal**: used to remove black borders at the edges of the image. Borders usually happen if the paper is smaller than the capture area, but smaller ones can occur if the paper was loaded slightly off.

- **Hole punch removal**: used to remove black circles in the margin that occur when scanning in paper with holes. If they aren't removed, they can interfere with OCR.

- **Blank page removal**: used to remove blank pages from a document. Especially useful if single-sided and double-sided documents are read with the same process.

- **Line removal**: removes form lines that might interfere with OCR.

- **Noise removal**: removes pixels that are the result of randomness introduced by either the capture device or thresholding.

- **Morphological Close**: closes small white holes in black areas. Especially useful to clean barcodes that have white noise in them.

- **Color Detection**: useful when scanning all documents as color and want to determine if the image actually contains useful color information. If color is not detected, the document can be thresholded to decrease the storage size without worrying about losing important color information. Good algorithms can differentiate between rich colorful photographs and single color markings (like highlights or red marks).

- **Drop-out color removal:** Some forms are designed so that parts of them can be rendered invisible during extraction. You often see this on bubble answer sheets where the bubbles might be a light red. Unfilled bubbles can be completely removed from the image when the form is scanned so that they aren't mistakenly recognized as a filled one.

Read barcodes

Barcodes on the document usually require third-party software to detect and decode, and if present, offer many possible benefits:

1. Some simple barcodes, like Patch, **can be used to signify a place to split documents**, allowing high-speed scanning of multiple documents at once.

2. Postal barcodes, like Postnet, encode **addresses, which can be put into other fields** in the database record.

3. Some barcodes, like Code 39, encode small strings which can refer to the ID of a database record and are **useful for relating data to this document** in your repository.

4. 2D barcodes, like PDF417 or DataMatrix, are used to **encode information about the document**. Decoding them can be a more accurate way to read the information over OCR.

5. Barcodes encode more than data, they can also be used to accurately **determine if the document is rotated**, so that you can correct that automatically.

Extract Metadata

The capture device provides more than just the image pixel data. Along with it, you get metadata describing the image. Storing metadata in separate fields that are related to the image in your repository will help you filter, sort or find this image later.

Some commonly available metadata fields that are useful to store separately

1. **Height and width**: very useful to know when viewing the image, but also useful if you want to filter photographs or CAD drawings by size.

2. **Number of pages**: also useful when viewing and can be used to filter for short or long documents.

3. **Color information**: knowing if the document contains color can help you pick the format to store it in.

4. **Thumbnails**: pre-made thumbnails are sometimes available and can help augment search results.

5. **Capture device data**: mostly useful for digital cameras that can report the original conditions that the photo was taken under and whether the flash went off.

6. **Location**: available on photos taken with cameras with a GPS.

OCR the document

Optical Character Recognition (OCR) algorithms are very useful for a variety of information extraction applications. Once you have the text, there are many uses for it:

1. **Put the text in a field** in your database related to the image, so that you can use full-text search capabilities of your repository to find the image.
2. **Create a Searchable PDF**, which is a PDF that has the recognized text accurately placed behind the original image. Standard search engines know how to index these files, and Acrobat Reader will allow you to search for, select, and copy the text out of the document.
3. **Parse the text**, and use it to populate separate fields. This is useful for forms processing, where you can pull out individual fields of a form.
4. **Detect if the page is rotated**, and correct it.
5. Use the text to **route the document.** For example, incoming faxes can be OCR'd and then emailed to person who is most likely addressed or responsible.
6. **Assist a human** in filling out a form. Using OCR and an interactive GUI, allow the user to select parts of the document to be recognized and use the text to auto-fill a form.

OMR the document

Optical Mark Recognition (OMR) can be used to find out how people have filled out standard forms that have checkboxes or bubbles to be filled in. Once the marks are recognized, the information can be used to route the form or fill out records in your database. There are five basic steps to OMR.

1. **Design the form**
2. **Create a template** for the marked areas
3. Perform transformations on the scanned image to **align and size it correctly** to match the template
4. Use image processing to **accentuate the marks**
5. **Find the marks** using the template

There are standard forms that have devices and software that have been built to read them. However, if you find that you have to do this yourself, you might want to use a simple barcode on the form (like Code 39) which is easy to recognize and can be used to align the form perfectly with your template.

Once you know which areas of the form to look for marks, recognizing them is usually done by taking a histogram of the brightness of the area and having a darkness threshold that indicates that the area was filled in. Doing this well requires the capture device to consistently render these images.

To make it more likely that you'll recognize marks, many forms use a drop-out color (like light red) that people can easily read on the form, but is easy to remove in pre-processing.

Human intervention

Image processing is not perfect, and so it's useful to make sure to have some human intervention. In chapter 2, we learned about how to make web applications that view enterprise images. Using those techniques, it's easy to involve people in image processing operations by building distributed web-deployed imaging applications. Humans can aid in information extraction by:

1. **Checking scans** to make sure that they are of good enough quality, and rescanning pages that aren't. (See chapter 3 for how to do this on the web)
2. **Checking the results of cleaning** operations didn't remove important parts of the document.
3. **Hand-tuning image processing** operations on images that weren't captured cleanly and might be especially challenging to clean automatically.
4. **Selecting regions of interest** in a document to focus OCR.
5. **Indexing the image** by choosing good tags and keywords that will help find the image later.

Sponsor

This e-course is sponsored by Atalasoft, which publishes a .NET Imaging SDK called DotImage, which can clean your documents, read barcodes, OCR images and create searchable PDFs.

Atalasoft's Five-Day Imaging e-Course

This is part five of five of Atalasoft's introduction to Enterprise Imaging. This five day course will cover the following topics:

1. Understanding Image Formats
2. Enterprise Imaging on the Web
3. Scanning from the Web
4. Extracting Information from Images
5. **Performance Tuning**

Chapter 5: Performance Tuning

Enterprise Imaging applications can be challenging to run efficiently. Unlike other data, documents and images are usually large, which means they take up a lot of memory, use a lot of disk storage, and take a long time to process or send over a network.

However, advanced image processing techniques can easily get you an order of magnitude improvement in size, speed or bandwidth. If you start using a few of these techniques, you'll see how easily you can reduce your hardware budget (which, incidentally, will reduce power consumption, maintenance, downtime, etc.)

Simply put, images are large. A 300 DPI color scan of a US Letter sized document uses over 25MB of memory when loaded. To process it, you need to look at over eight million pixels. By comparison, the complete works of Shakespeare take up about 5MB of text data.

Hard-disk, Memory, Bandwidth and CPU power are getting cheaper all of the time. However, if you want to keep up with a high-speed scanner, process the documents intelligently, service multiple clients by processing images on a server, and keep a complete archive (and backups), then you will need to know how to scale. A one-time cost of optimizing your software can save you money now and in future maintenance costs.

How to reduce memory usage

The memory needed by an image in memory is mostly determined by the formula:

Pixel Width x **Pixel Height** x **Bits per pixel** / 8

Reducing any of those numbers will reduce the memory needed to render or process the image. However, reducing any of those numbers may also reduce the quality.

Tip 1: ***Resample the image to a smaller size*** and adjust the DPI so that it prints to the same size. A color scan of US letter size paper at 300 DPI is 2,550 pixels wide by 3,300 pixels long, for a total of 25MB. If you resample so that you cut each dimension in half, and then adjust the DPI to 150, your image takes up just over 6MB, or 25% of the original.

Tip 2: ***Convert to grayscale or black and white.*** If your document is using 24-bit color, but you don't mind losing color, you can convert to grayscale, which uses about 33% of the space. If you can convert to 1-bit without losing meaning, your documents will be about 4% of the original size.

Tip 3: ***Reduce the number of colors and use an indexed format.*** Images with only a few color regions (such as a document with highlights and other color markings) can usually be converted to an indexed format without losing quality. You can choose 8-bit indexed if you have up to 256 colors or 4-bit indexed if you have up to 16 colors. Remember, black and white and each level of gray you need count as colors. Images of this type are sometimes called paletted or color-mapped images.

Tip 4: ***Process pages one at a time.*** If you use a multipage format such as TIFF or PDF, do not load all frames into memory at the same time. If you need to process them all, load one frame at a time, and then use the appending features of your codec to build up an output file. This is especially important if you don't know how many pages you are going to process.

Tip 5: ***Use codecs that can read regions of images efficiently.*** Some formats, like TIFF, can store images as tiles or strips. If you only need a portion of the image, make sure to use your codec's read region facility to extract just the pixels you need. You should not read the whole image and then crop to size.

Tip 6: ***Use automated border crop.*** Some scans, especially of smaller items, like checks, have a large dark border around the edges. Use an algorithm that can detect and remove this, leaving you with just the important part of the image. Incidentally, this will save you ink if you print these documents.

How to reduce CPU usage

Tip 7: ***Reducing size reduces CPU usage.*** Most image processing commands need to look at every pixel of an image to do their job. Having fewer and smaller pixels means they'll be processed faster.

Tip 8: ***Take advantage of multi-core processing.*** Modern computers have multiple processors, each having multiple cores. You only get this advantage if your software is written to take advantage of it. If you are processing a lot of images at once, set up queues and use worker threads to process them. If you are processing very large images one at a time, then make sure you use image processing algorithms that: cut the images into tiles, spawn threads to process them, and then stitch them back together at the end. Not every image processing algorithm can be written this way, but many common ones, like converting to black and white, can.

How to reduce I/O time

Tip 9: ***Make a local copy of remote images.*** Most image formats were designed to require random access to the image store. This means that when you read in images, you may need to use many different parts of the file, in no particular order. If the device that contains your images has expensive seek times (like a network share) then you will benefit by making a local copy first, and then reading from that.

Tip 10: ***Get information about your images once.*** Don't keep reading the stream to find out height, width or number of frames. Cache the results, because getting them again and again can be costly. If you store information about your images outside the image (like in a database row), caching some commonly needed information there can save you a lot of time.

Tip 11: ***If you know the image format, don't use auto-discovery algorithms.*** If you already know what the format of the image is before you open it, then use the specific codec instead of trying to use an auto-discovery algorithm. Each codec will independently read the image file to try to figure out the format, causing many unnecessary file accesses.

Tip 12: *Use tiled formats.* If you often need just part of an image, use a tiled format, such as Tiled-TIFF, which makes getting regions of the image faster.

How to reduce storage size

It's hard to predict what the final file size of an image will be, but generally, it's proportional to:

Pixel Width x **Pixel Height** x **Bits per pixel** x **Number of Frames + Metadata**

It's more complicated if each frame uses a different number of bits per pixel (then you need to add up each image's size), and some formats can store metadata for each frame. But, generally, reducing any of those numbers will reduce the space needed to store the image.

Tip 13: *Use techniques to reduce memory usage*. The formula above is a lot like the formula for memory. That means if you use a technique to reduce your image's memory usage then you are likely to reduce the space needed to store it.

Tip 14: *Detect color and use a layered format.* If your documents are coming in off a color scanner and are a mix of color and black and white, use a color-detection and segmentation algorithm to make layers. In a PDF, the color layers can be placed on top of the black and white layers. This works extremely well with documents like magazines, which are mostly black and white text with rectangular color images and ads.

Tip 15: *Use a better compression algorithm.* Advanced compression algorithms like JBIG2 and JPEG2000 can result in smaller files without sacrificing quality.

Tip 16: *Use a lossy compression algorithm.* Lossy formats like JPEG allow you to lower the quality in exchange for smaller files.

Tip 17: *Remove blank pages.* If you are scanning two-sided documents, you probably have some blank pages. Detect and remove them.

Tip 18: *Remove unneeded metadata.* Images often carry around extra metadata that was put in by the device or software that created them. If you don't need it, remove it.

How to reduce bandwidth

Tip 19: *Reducing storage size reduces bandwidth*. Since images are usually sent over the network in their compressed form, reducing an image's storage size will use less bandwidth. Sometimes a technique, like removing metadata or reducing quality, is not acceptable for the archived format, but is ok right before sending it over the network.

Tip 20: *Create thumbnails on the server, and send them on demand.* If you are preparing a web page of thumbnails, then make them on the server (don't use browser features to resize them). Use JavaScript to detect if the thumbnail is on the screen, and request it on demand.

Tip 21: *Tile the image, and only send what you need.* If you have a large format image, such as a scanned mechanical drawing, send only the visible parts to the browser and send the rest if the user scrolls over to it. Even doing this for letter sized scans can save you a lot of time and bandwidth.

Tradeoffs

Optimizations often have some tradeoff. This table will help you choose the methods that are right for you.

Tip	Reduces	Possible Tradeoffs
Resampling	Memory, storage, CPU, Bandwidth	Quality and up-front CPU time
Converting to gray	Memory, storage, CPU, Bandwidth	Quality and up-front CPU time
Reducing colors	Memory, storage, CPU, Bandwidth	Some image processing algorithms won't work, and that will require you to convert back to a non-indexed (or continuous) format.
Processing one page at a time	Memory, View Time	Possible increased storage time.
Reading regions	Memory, I/O Time	If the regions are nearly the same as the image or not matched to the file format's stored regions, it could take longer to read them.
Cropping borders	Memory, Storage, CPU, Bandwidth	If the algorithm gets the border wrong, you'll lose part of your image. May need to add a QA process.
Multicore processing	CPU	Has fixed CPU overhead, so only useful if the savings can overcome that.
Making local copies	I/O Time	One time cost to make the local copy may not be offset if you don't read much of the file.
Caching information	I/O Time	Uses memory and storage space.
Using specific codecs	I/O Time	Only useful if you know the format.
Using tiled formats	I/O Time	Usually results in slightly bigger files, and takes longer to read and write the whole image.
Detecting color	Storage, Bandwidth	May take time – make sure the algorithm can keep up with your throughput. Loss of color if the algorithm gets it wrong – may need to add a QA process.
Using a better compression	Storage, Bandwidth	May not be supported by all of the software you need to read the image.
Use a Lossy Compression Algorithm	Storage, Bandwidth	Reduces the quality (obviously), but this can get progressively worse if you need to edit the image multiple times, and resave it.
Removing blank pages	Storage, Bandwidth	Only a problem if the algorithm incorrectly removes a non-blank page. May need to add a QA process.
Removing metadata	Storage, Bandwidth	Ok, as long as you will never need to recover it.
Thumbnailing on the server on-demand	Bandwidth, View time	Thumbnails not there already when the user scrolls. If you need to increase the thumbnail size on the fly, you need to keep resending them. Uses more server CPU.
Tiling the image before sending it	Bandwidth, View time	Tiles aren't there when the user scrolls. Remember to cache tiles, or else you'll send them again when the user scrolls back. Uses more server CPU.

Sponsor

This e-Course is sponsored by Atalasoft, which publishes a .NET Imaging SDK called DotImage, which implements the tips found in this chapter (most of them automatically).

BONUS Chapter:

Understanding TIFF

Introduction

This document is a description of the TIFF image format. After reading this description, you should have a working understanding of how the files are laid out and why. It should also give you a greater understanding of the challenges of reading and writing these files correctly and how to investigate problems that some files may have.

TIFF, or Tagged Image File Format, is a specification for representing images and associated data. The TIFF standard was originally published by Aldus (later acquired by Adobe) in 1986. As a file format, TIFF had several important goals [not available in their entirety] in other file formats:

- The format should be reasonably easy to generate
- The format should be usable on many platforms
- The format should be usable in low-memory conditions
- The format should be able to hold many images
- The format should be able to be generated using limited processing power
- The format should be able to store extra information about the image (i.e., metadata)
- The format should allow quick updating/rearranging of pages

Structure

TIFF meets all of the goals, to a certain degree. Programs that generated TIFF files were in use on computers that had little more than one megabyte of RAM and didn't have virtual memory. Such systems also were very slow by today's standards.

A TIFF file consists of three types of main components: a **header**, image file directories (**IFD**) and raw **data**. The header identifies the file as TIFF. The IFDs specify image information and other data. Raw data is special purpose information within the file. Each of these elements is linked to other elements. The header links to the first IFD, each IFD links to the next IFD, and each IFD links to the data that it uses.

The TIFF specification was written to best support small platforms generating TIFF files. To this end, Aldus recognized that it requires less code to write data in the same format used by a particular processor. Some data elements, such as short integers, long integers, or double precision floating point numbers require more than one byte to represent them. Short integers, for example, require two bytes to represent them. Consider the number 1025. As a short integer, this is represented by two bytes, 4 and 1 (4 x 256 + 1 = 1025). The 1 is often referred to as the low byte and the 4 as the high byte. Borrowing from Gulliver's Travels, this is called an **endian** problem. When writing out 1025, which byte gets written first - the little end or the big end? For TIFF, the answer is "whatever suits your platform the best."

At the time that the TIFF spec was written, Motorola and Intel had processors in many of the major desktop publishing platforms. Motorola processors typically used big endian numbers (the high byte comes first in memory). Intel processors typically used little endian numbers (the low byte comes first in memory).

The Header

The TIFF header starts with a two byte platform ID which is either II or MM. If it is II, then the data in the file is in Intel byte order (little endian). If it is MM, then the data in the file is in Motorola byte order (big endian). Following this is a two byte short integer "magic number", 42, which is supposed to identify the file as TIFF. Finally, there is four byte long integer, which identifies where the first IFD is located within the file.

TIFF Header	
Element	**Description**
Byte order	Two bytes, either II or MM.
Magic Number	Two byte short integer, 42
First Ifd	Four byte long integer, describing the offset from the beginning of the file to the first IFD. The spec requires this to be an even number.

From the point of view of a 1986 software developer, this kind of flexibility in the specification is a big win. The header and the IFDs can often be written out with one or two lines of code. The downside is the reading arbitrary TIFFs is more complicated.

IFDs

For every image within a TIFF file, there will be an IFD that represents it. An IFD is a collection of tags that represent specific information. For example, there is a tag that represents the height of the image and a tag that represents the width of the image. Each IFD starts with a short integer that represents how many tags are in the IFD, then there are the tags themselves, then a long integer that represents where to find the next IFD in the file. If this last value is 0, then there are no more IFDs in the file.

IFD	
Element	**Description**
Tag Count	Two bytes short integer
Tags	Tag Count * 12 bytes of data representing tags
Next IFD	Four byte long integer, describing the offset from the beginning of the file to the first IFD. The spec requires this to be an even number.

While there must be an IFD for every image in the file, not every IFD needs to be an image. Since an IFD is simply a collection of tags, the information may not have anything to do with an image. Typically though, an IFD is used for describing an image. Let's take a moment to solve the problem of how many pages are contained in a TIFF.

Given what we know so far, we can get an answer.

1. Read the header and get the offset to the first IFD
2. Set page count to zero
3. If the offset is zero, stop
4. Skip to the offset and read the tag count
5. Add one to page count
6. Skip 12 bytes times the tag count forward and read the offset to the next IFD
7. Repeat steps 3-6 as needed

This process will end up counting all the IFDs in the file. There are a few things missing, though. Since an IFD may not be an image IFD, step 5 should instead be "Examine tags, and if this IFD is an image IFD, add one to page count." In addition, since many different applications are written by many different authors, there may be unusual interpretations of the TIFF specification. The specification says that "Readers must follow the [IFD offset] pointers wherever they may lead." Some TIFF files have offset pointers that point beyond the end of file. This condition can also mean that there are no more IFDs past the current. In addition, files may be truncated or otherwise damaged, and readers should cope with that as well.

In addition, IFDs need not be in order within the file. It is OK for the first IFD to be physically at the end of the file. Therefore, pages in a TIFF file can be reordered by just changing the offsets of the IFDs. Pages can be inserted by adding them to the end and changing the IFD offsets. Pages can be removed by unhooking them from the chain of IFDs – but this doesn't allow the space they take up to be reclaimed. Permanently removing a page is a much more complicated process.

Tags

Tags within IFDs are 12 byte blocks. Every **tag** consists of four elements: a two byte ID, a two byte data type, a four byte count, and a four byte value. The ID describes what the meaning of the tag should be. This includes image dimensions, resolution, bit depth, colors, and so on. The TIFF spec describes a set of IDs that are considered "baseline" in that all readers should understand their meaning. The data type describes how the value should be interpreted. It can mean byte, short integer, long integer, rational (i.e. fractions), floating points numbers, unformatted data, etc. The count indicates how many elements of the data type are in the file. This is how tags can describes arrays or strings (strings are arrays of bytes).

Finally there are four bytes for storing the actual value. If the value itself can fit in four bytes, it will be put into the value. If it can't, the value is interpreted as an offset to the actual value in the file. For example, a long integer can fit into the value field so if the count field is 1, then the value field will contain the literal value. If the count field is greater than 1, then the value field is an offset to where the values will be found.

TIFF Tag	
Element	**Description**
ID	Two byte short integer representing the tag's meaning
Type	Two byte short integer representing the type of the tag's value
Count	The number of value elements. 1 means a single value, > 1 means an array
Value/Offset	Four byte value, either the literal value of the tag or an offset from the beginning of the file to where the value can be found.

Image Types

There are four general image types that can be represented in a TIFF file: bi-level (usually black and white), gray, paletted, and full color. Image IFDs are required to have tags that represent the image's width, height, compression, resolution, image data, and photometric interpretation. Photometric interpretation describes the meaning of a zero for image data. It can mean either no value (usually black) or full value (usually white). It may also indicate that the image is color or paletted or transparent. Even though the photometric interpretation tag is required, many IFDs are missing it, requiring TIFF readers to infer the type of image. In addition to common image types, TIFF images may use more esoteric color representations, including Lab, YCbCr, and CMYK.

Image Data

Image data within a TIFF can be stored in three fundamental ways: a single block, a series of strips, or a series of tiles. In the case of a single block, the image data is all in one place, sequentially. Storing an image this way allows for very efficient reading and writing as well as better compression. It makes it more challenging to read a region of interest (ROI) of the image as all image data up to the region would need to be discarded as it is read. To handle these cases, TIFF allows producers to create images that are built from a series of strips or a series of tiles. This allows better ROI access, but these types of files are harder to create and usually take up more space due to compression overhead being present on every strip or tile. When an image is comprised of strips, the strips are all the same height, except for the last which may be smaller. When an image is comprised of tiles, all tiles are the same size, whether or not the image exactly covers all the tiles. Tiles do not need to be square, but the width must be a multiple of 16.

Compression

Data compression may be applied to the image data before writing it out. In the case of stripped or tiled images, each strip or tile is compressed individually, but the same compression will be used for each tile.

TIFF allows the author to select the most appropriate compression. This can include CCITT (fax) compression for black and white images only, Flate/LZW compression, JPEG, and Run Length Encoding. Each compression has its own set of gains and losses depending on image type and content.

Metadata

TIFF files may also contain metadata that describes the image in greater detail. Metadata can be stored directly in tags within an IFD (author, copyright, host computer, etc), or it may in a separate collection. One such collection is called EXIF. It is another collection of standard tags and values and includes a wider variety of information about the image, including the make/model of the imaging device, the white balance, the settings of the imaging device (aperture, focal length, exposure time) and the location of where the image was taken in GPS coordinates.

Conclusion

While there is a great deal of flexibility in the TIFF file format, the same flexibility has created a set of problems. Many TIFF creation programs routinely create non-compliant TIFF files which are not uniformly readable or interpreted correctly by all programs and some TIFF reading programs do not correctly interpret the specification when reading otherwise compliant files. As a result, there are a number of files available "in the wild" that can be very challenging to read.

Sponsor

This e-Course is sponsored by Atalasoft, which publishes a .NET Imaging SDK called DotImage, which implements the tips found in this chapter (most of them automatically).
Take a look at http://www.atalasoft.com/products/dotimage