


Summary of Comments on PDFTutorial050411Annotated.pdf

Page: 1

Author: jameskin

Subject: Note

Date: 5/1/05 12:53 PM

 Welcome to our look inside PDF. We will literally be looking inside a set of example PDF files. We will be going over a lot of details but the real objective is to let you carry away a general understanding of what kinds of things are inside of PDF files and how they are organized and relate to one another. So don't let the details discourage you.

The actual example files are a companion to this slide presentation. You can view the example files with Adobe Reader or Acrobat Standard or Professional.

You can also look inside the files yourself by using any text editor like Notepad, Microsoft Word, Emacs, etc. If you really get into it you can make small changes to the files and then display them with Acrobat to see the effect.

Well hold on, here we go,


Jim King -- 4/26/2005

Page: 2

Author: jameskin

Subject: Note

Date: 4/30/05 11:49 AM

 In June 1993 Adobe announced Acrobat and the first documentation on PDF was available from Addison Wesley Publishing Company called the Portable Document Format Language Reference Manual. Those books are still for sale and a new one just came out called the 5th Edition.

That 5th Edition documents PDF version 1.6 and it can be found in electronic format on Adobe's World Wide Web site as noted on the slide.

Information about the archiving subset of PDF, PDF/A can be found on the AIIM site as shown.


This presentation together with the example PDF files will be on my private Web site (Comcast) as shown. You will find several other tutorial presentations of mine at that site as well.

Page: 3

Author: jameskin

Subject: Note

Date: 4/30/05 11:49 AM


 More and more I am beginning to look at PDF files as composite documents. PDFs are self contained complete documents that are made from images, graphics, text blocks, and have document features like table of contents, digital signatures, etc.

Page: 4

Author: jameskin

Subject: Note

Date: 4/30/05 11:48 AM

 These are just two examples to display what I mean by composite documents.

The one of the left has several images down the left hand side. I has a company graphic at the top right and it has nicely laid out text.

The right one has a complicated graphic, a logo and a form to fill out on the bottom.


Composite docments are made from distinct document pieces of different types.

Page: 5

Author: jameskin

Subject: Note

Date: 4/30/05 11:48 AM


 Now we step back and go through the basics of exactly how a PDF file is put together. It is really very simple, yet very powerful.

Page: 6

Author: jameskin

Subject: Note

Date: 4/30/05 11:48 AM


 This slide is pretty self describing if you read it from top to bottom. We will see exactly what objects look like in the next 50 slides or so.

Page: 7

Author: jameskin

Subject: Note

Date: 4/30/05 11:47 AM

 This is a sample page that is a take off on the sample programs people write to output "Hello World". Here we have a page (8.5 x 6.19 inches) laid out to match the format of the rest of the slides.

We might more normally in the US have a page that is 8.5 x 11 inches. We will see how the page size is determined.


The words "Hello World" are presented roughly in the middle of the page in 24 point Helvetica.

Page: 8

Author: jameskin

Subject: Note

Date: 4/30/05 11:47 AM

 We will see a lot of pages like this one so let me tell you carefully what you are looking at. I opened the example 01 PDF file in Microsoft Word as a "text" file. The characters from the PDF file have been formatted into three columns and I have also inserted some line breaks and indentations in order to make the text more readable. I also added headings and footings including page numbers.

There are sections highlighted in colors so that your attention is drawn to the parts we will be discussing. If you do a text copy from this page and put it into a simple text page, then save it as a file, that file will be a PDF document that will display Hello World. If you really try this with Word, make sure that you do a "saveas" and select "text only with line breaks" as the format. Do not expect this to work if you save it as a Word document.


Since the page displays "Hello World" you might expect the PDF file to have that character string somewhere within it. You would be right and I have highlighted this in red. We will work our way outward from this string and see what supporting material is required to turn that text string into a complete PDF document. Notice that "Hello World" is enclosed in parenthesis. This is how strings are represented in both PDF and PostScript.

Page: 9

Author: jameskin

Subject: Note

Date: 4/30/05 11:47 AM

 Note that the material in the file is organized into 6 objects a "%PDF-1.2" header and a trailer.

Each of the objects has a number followed by a zero, begins with "obj" and ends with "endobj".

I have often referred to PDF as "object oriented PostScript" because this object structuring is something that PDF has but PostScript does not.


Note also that the file starts with "%PDF-1.2" which indicates that this is a PDF file following the 1.2 version of the PDF specification. Since the current products support PDF 1.6 I could have put that on the first line. But there is nothing in this file that was not supported by the very early products.

Page: 10

Author: jameskin

Subject: Note

Date: 4/30/05 11:51 AM

 Objects are numbered and the second numbers shown here as zeroes are the object's version numbers. This is because it is possible to update PDF files "inplace" without copying the whole file and sometimes it is necessary to indicate that this is a new version of a particular object. So object "1 2" would be a newer version of object 1 than would be "1 0".

I have highlighted two ways in which the object numbers are used. The simple arrowheads indicate the object definitions whereas the full arrows show where references to the object are found within other objects. The "R" is a notation that indicates that the two preceding numbers form a reference to an object.


The PDF document actually starts at the end of the file at the "trailer" where we see that the "/" Root" of the document is object 6. If we look at object 6 we see that it is the document's "/" Catalog" and it refers to the document's "/"Pages" as object 5. Object 5 then has a reference to a single "/"Kids" page at object 1 which is of "/"Type" "/"Page".

Page: 11

Author: jameskin

Subject: Note

Date: 4/11/05 5:15 PM

 Here is a pictorial diagram of the initial structure required of all PDF files. The Root always points to the Catalog which points to the Pages node. The Pages node has one or more Kids nodes which represent the individual pages in the document.

Note that the Page nodes each point back to the Pages node. Each Page node will subsequently point further to other objects that make up each page. We will be looking at those objects a little later on.


When we use the term "points to" or "refers to" we mean the linking established by the object numbers and references to them from within other objects.

Page: 12

Author: jameskin

Subject: Note

Date: 4/30/05 11:47 AM

 Before we get bogged down in more details of what is in a PDF file, let's look at another simple example (02).


This is the same "Hello World" page but if you look closely you will see that the words are only 50% gray, not completely black. You might want to zoom in to take a closer look.

Page: 13

Author: jameskin

Subject: Note

Date: 4/30/05 11:46 AM

 Again, no surprises. There is one new thing in this file that wasn't in the first: "0.5 g" which stands for 50% gray.

The default color is set to 100% gray or black at the start of each page. As long as black is what is wanted no further color settings are needed as was the case in the first example.


Note an important point about PDF. The setting of things like color and fonts are reset to the defaults at the start of each page. Settings on one page have no effect on setting on other pages. This gives up page independence, something that PostScript doesn't do.

Page: 14

Author: jameskin

Subject: Note

Date: 4/11/05 5:18 PM

 Well, we will be look at more and more of the constructs found in the PDF files so let us stop for a minute and talk about the notation that is being used.

Both PostScript and PDF use a notation known as "postfix". (That is where the name PostScript came from.) Postfix is an traditional mathematical notation sometimes call reverse "Polish" notation. The unique thing about this notation is that the "action" indicator, more often called the "operator", comes at the right hand end of the expression and complex expressions can be written without the use of parenthesis. There is also a notation known as "prefix" notation where the operator comes first. The normal notation we usually use (e.g., $a + b$) is called "infix" notation because the operator is between or within the expression.

Postfix notation is sometimes used by hand calculators.

We have already seen the "0.5 g" expression where the number 0.5 is the first and only operand and the operator is "g". We have also seen the text string which is indicated by enclosing the characters in parenthesis which becomes the single operand to the operator "Tj". The "T" in this operator must be capitalized and it stands for "text". PostScript also uses 0.5 g but instead of "Tj" PostScript uses "show". So the example in PostScript would look like "(Hello World) show". We have also seen the object reference operator "R" which takes the object number and version number as its two preceding operands. I have come to look at "5 0 R" as one single thing -- a pointer off to object 5.

Page: 15

Author: jameskin

Subject: Note

Date: 4/30/05 11:46 AM

 Continuing to work our way up from the "Hello World" string, we find the operator "Tm". This stands for "text matrix" and takes the six preceding numbers as its arguments.

Those six numbers allow us to indicate scaling, skewing, rotating and moving of the material on the page or drawing surface.


Note that the last two numbers are 260 and 254. This indicates where on the page we want the Hello World string to occur. The first four numbers (1 0 0 1) indicate that no rotation, scaling or skewing is to occur.

Page: 16

Author: jameskin

Subject: Note

Date: 4/11/05 5:20 PM


 Make a mental note of where on the page the Hello World string is placed. In particular think of the point just to the left of the "H" of Hello and at the characters baseline. Think of this position in relation to the bottom left corner of the drawing surface.

Page: 17

Author: jameskin

Subject: Note

Date: 4/11/05 5:22 PM

 The default measurement system in both PostScript and PDF is in units of 1/72 of an inch and the starting point is in the lower left corner of the page or drawing area.

So for an 8.5 by 11 inch page there are 612 units horizontally and 792 units vertically. Think of graph paper.

We are using pages that are 8.5 by 6.19 inches.

This means that the two numbers we saw in the "Tm" operation indicate the horizontal (X) offset from the left edge of the page (260) and the vertical (Y) offset from the bottom (254). So our string is to be placed 260/612 -ths in from the left and 254/466 -ths up from the bottom.


The number 446 is what you get rounded from multiplying 6.19 inches by 72 units to the inch.

Page: 18

Author: jameskin

Subject: Note

Date: 4/11/05 5:22 PM


 Again, working up from "Hello World" we encounter the "Tf" operator which appears to have two preceding arguments "/F1" an arbitrary name and "24" a number. "Tf" stands for "text font" and so it is safe to assume that the 24 means the point size. But what about the "/F1" which we said was an arbitrary "name"?

Page: 19

Author: jameskin

Subject: Note

Date: 4/11/05 5:24 PM

 Well, lets go back to some more basics. This time we look at the building blocks of all PDF files. Most of these constructs are also used in PostScript. You will learn quite a lot about PostScript here if you are not already familiar with it.

We mentioned strings already -- contained within parenthesis. There are several more rules for forming strings like what do you do when you need to show parenthesis. We leave those details to the be looked up in the PDF Reference Manual.

Numbers are strings of digits. In PDF and PostScript they can have decimal points. In PostScript one can also use scientific notation like 1.5E-06, but that is not allowed in PDF.

Names or labels always begin with a "/" (slash) character. They are used as "keys" in dictionaries and in various ways to denote or label concepts and things.

Arrays are ordered, numbered, lists of things and always begin with a "[" (left square bracket) and end with a matching "]" (right square bracket). Between the brackets are the array elements separated by white space. The elements can be of any type, mixed or not. That is, PDF and PostScript support "heterogeneous" arrays. The elements are implicitly numbered from left to right starting with zero.

One of the most used and most powerful constructs used in PDF (and also in PostScript) is the "dictionary". These begin with "<<" (two less thans) and end with a matching ">>" (two greater thans). In between them are pairs of items, the first of each pair being a name. The second element of each pair can be any single construct. The first of each pair is the "key" and the second is the "value". So dictionaries provide name/value pairs, associative stores, or whatever other term you might have heard of where you can save and look up values by a key or name.


We already mentioned the "references" which use the postfix operator "R" and have the object number and version as the preceding arguments.

Page: 20

Author: jameskin

Subject: Note

Date: 4/11/05 5:25 PM

 The basic elements shown on the preceding page can be put together to form more complicated structures by nesting elements inside one another.

The first example is an array that contains three items. Item zero is a dictionary, item 1 is 22 and item 2 is 44.55. The dictionary itself contains three entries. The first is the Name which has a string (Jim) as a value, Age which has the number 39 as a value and Children which has an array of the three strings (Heather) (Timothy) and (Rebecca) the names of my three children. And I'm 39 too!

The next example is a dictionary containing three entries, MORE which has as value an array of 5 numbers, LESS which has as value an array of three strings, and count which has a value of 88.


Note that upper and lower case letters are always significant in PDF and PostScript. So "More" is different from "MORE" and "count" is not the same as "Count".

Page: 21

Author: jameskin

Subject: Note

Date: 4/11/05 5:27 PM

 We have already talked some about PDF objects. This is a construct that the PostScript Language does NOT have. The ability to randomly access portions of PDF a document and the page independence of PDF pages are based on the object structuring of PDF.

Objects always begin with the "obj" operator which takes two preceding numbers and ends with the "endobj" (end object) operator. In between, can be any one of the previously noted elements: a number, a string, a name, a reference, a dictionary or an array. On the next page we will see another new PDF construct, the "stream" that can also be within an object.

Notice that once objects and references to objects have been introduced, in many cases, there are alternative ways in which to represent more complex structures either using nesting with what are called "direct" objects or remotely via "indirect" objects. This is shown where the dictionary can contain the string (a labrador) nested within it or alternatively could nest only the reference to the string "4 0 R" and the string becomes an independent and indirect object somewhere else in the file.


The power of indirect objects is that they can be shared. That is, more than one reference can refer to any given object. If the object is large this can be a great space saver over repeating the object within each containing (referencing) object.

Page: 22

Author: jameskin

Subject: Note

Date: 4/11/05 5:27 PM

 The last construct we will talk about is the "stream". Streams are objects containing a dictionary and a stream of material contained between the delimiters "stream" and "endstream". The dictionary must contain the key "/Length" followed by the stream length in bytes.

Streams are used in many ways in PDF documents. One important use is to hold sampled image data. Images sampled at very high resolutions can be extremely large (e.g., 50 megabytes). The bytes don't fit the model for a simple string, array, etc. The stream is the place where potentially large and perhaps unstructured material or material requiring a structure different from the basic object structure can be placed.


This construct allows implementations to have two distinct methods for obtaining material out of the file. The objects are usually relatively small and can be read in their entirety. Streams are the place to put data that doesn't fit this model.

Page: 23

Author: jameskin

Subject: Note

Date: 4/11/05 5:30 PM

 We now return to our 50% gray "Hello World" example after having learned more about the details of the PDF notation.

We had been working our way up from the Hello World string and had arrived at the "Tf" operator which takes two preceding operands. The 24 is the font point size. The /F1 is an arbitrary made-up name for the font. Having the full long font names within the page content repeated every time there is a font change would be pretty bulky. So the design allow for a made up short font name (e.g., /F1) to be used to refer off to a "font resource". All resource are referenced via a "resource dictionary" that is associated with each page.

The page content object number 2 is pointed to by the /Page object number 1 by means of the /Contents key. You will notice that the /Page object also has a /Resources key which, in this case, references object 3. If we then look at object 3 we see that it has a key /Font which has as its value a dictionary with the made up name /F1 which then refers to object 4. Looking at object 4 we see that it is an object of /Type /Font with a /Subtype of /Type1 (the Adobe PostScript font format) and it also has a key indicating the font is a /Basefont and is that one named /Helvetica. Base Fonts are the base 13 fonts of PostScript and must be built-in to any PDF processing program. In this way the font definition need no further information to define the font being used. In the case of fonts not built-in, this font object is more complex and points off to character with tables, encoding vectors, and the actual font outlines.

Other types supported besides /Type1 include /TrueType and /Type3 and subsets of them.

Let us finish off this page content object completely. We have discussed the operators "Tj", "g", "Tm" and "Tf". That leaves the "BT" or begin text and the "ET" or end text operator. This is another place where PDF differs from PostScript. In PDF the BT and ET form a text block which is not allowed to contain graphics or image operators. It was felt that being able to find and isolate text blocks would help in reading and updating PDF files. PostScript does not have this segregation of text -- all imaging operators are at the same level.


That completes the description of all of the content operators within this page contents stream. Streams were discussed earlier. The contents of each page is held within a stream as in this object 2.

Page: 24

Author: jameskin

Subject: Note

Date: 4/11/05 5:30 PM


 Onward to example 3! We see that our "Hello World" string has become red.

Page: 25

Author: jameskin

Subject: Note

Date: 4/11/05 5:31 PM

 We see the "rg" operator highlighted.

One of the design criteria for PDF was to make the files as small as possible. So one letter abbreviations were used for operator names. When most of the one letter suggestive abbreviations were exhausted the designers turned to two letter abbreviations. So "rg" is the abbreviation for "rgb" or red, green, blue.


This is a device dependent color request asking for 100% of the red colorant to be used and zero percent of the green and blue. The three arguments preceding the "rg" operator indicate the percentage of red, green and blue, respectively. Those numbers range from 0 (none) to 1 (100%) and can contain decimal points (e.g., 0.45 would indicate to use of 45% of that colorant).

Page: 26

Author: jameskin

Subject: Note

Date: 4/11/05 5:31 PM


 Another red "Hello World"?

Page: 27

Author: jameskin

Subject: Note

Date: 4/11/05 5:33 PM

 This time we show how device independent color designations can be used in PDF. There are two operators used in this example: "cs" which is the abbreviation for "set the color space" and "sc" which is the abbreviation for "set the color within the 'current' color space".

Looking first at the "cs" operator we see that it takes one preceding argument which is a made up name /CS1. This was a name of my choosing. It could just as easily been /mycolorspace or /X or whatever.

Color spaces are considered a resource just like fonts so they are also looked up in the resources dictionary -- in this example, object 3. We see that this dictionary contains an entry for /Colorspace which has a dictionary as its value. Within that dictionary we find the key /CS1 which has as its value the color space specification for our color space as an array. A more readable form of the colorspace definition is given on the next page.

The first element of the array is the name /Lab which defines this color space to be the CIE L*a*b* color space. The next element of the array is a dictionary that defines two keys /Range and /WhitePoint. The /Range key has as its value an array of four numbers giving the range of a* and b* values respectively. The value of L* is always assumed to be between 0 and 100..

The /WhitePoint key has as its value an array of 3 values providing the CIE XYZ values, respectively, of the diffuse white point of the color space.

The "ri" operator is next after the "cs" operator (back to the page content object 2) and it stands for "rendering intent". Those of you seriously into the color management topic will recognize this as determining the rendering intent of "AbsoluteColorimetric" which is one of 4 choices for how to do gamut compression and other adjustments. The single argument to "ri" is the name of a rendering intent.


Next the "sc" or set color operator has as its three preceding operands, the value of L*, a* and b*, respectively. Trust me, 63, 127, 127 is a red color in L*a*b*.

Page: 28

Author: jameskin

Subject: Note

Date: 4/11/05 5:34 PM

 Color spaces are considered a resource just like fonts so they are also looked up in the resources dictionary -- in this example object 3. We see that this dictionary contains an entry for /Colorspace which has a dictionary as its value. Within that dictionary we find the key /CS1 which has as its value the color space specification for our color space as an array.

The first element of the array is the name /Lab which defines this color space to be the CIE L*a*b* color space. The next element of the array is a dictionary that defines two keys /Range and /WhitePoint. The /Range key has as its value an array of four numbers giving the range of a* and b* values respectively. The value of L* is always assumed to be between 0 and 100..


The /WhitePoint key has as its value an array of 3 values providing the CIE XYZ values, respectively, of the diffuse white point of the color space.

Page: 29

Author: jameskin

Subject: Note

Date: 4/11/05 5:35 PM


 A third red "Hello World"?

Page: 30

Author: jameskin

Subject: Note

Date: 4/11/05 5:35 PM

 This time we also specify a red color using a device independent color space -- this one is a "calibrated RGB" color space.

Again, we have a made-up name for the color space /CS2 and it can be looked up in the resources dictionary -- object 3.

Since this color space is an rgb one, the "sc" set color operator has as its arguments three numbers for red, green, and blue, respectively.

In this case, red -- 100%, green -- 0%, and blue 0%.


To get a better formatted view of the /CS2 color space go to the next page.

Note that this example does not have any "ri" or rendering intent specified. In this case it will use the "default" setting which is for "RelativeColorimetric" processing.

Author: jameskin

Subject: Note

Date: 4/11/05 5:36 PM

 This is a "calibrated RGB" color space indicated by the name /CalRGB being the first element of the /CS2 array. The second element is a dictionary containing the properties that would distinguish this particular RGB color space from all other RGB color spaces.


The properties enumerated here are the gammas for each of red, green and blue, respectively (all 2.222 in this case) and the diffuse white point of the color space given as a CIE XYZ value. In addition, this definition has a /Matrix key whose value is an array of nine elements. These elements are three groups of three numbers and specify the transformation of an rgb value in this space, after gamma correction to the CIE XYZ standard. The first three numbers are for red, the next three for green and the last three for blue.

Page: 32

Author: jameskin

Subject: Note

Date: 4/11/05 5:36 PM

 Finally, no more red "Hello World"!

This time we have added a blue star to the black "Hello World" page of example 01.


You might want to zoom in to take a closer look at the star.

Page: 33

Author: jameskin

Subject: Note

Date: 4/11/05 5:38 PM

 The blue star is "drawn" using line graphics operators "l", "m" and "f". This is a very compact and expressive way to create lots of colored objects quickly. Graphics expressed this way also rotate, scale and move with no degradation and are completely resolution independent. This is a feature of PDF and PostScript and the degree to which it is precise and expressive is rather unique to these Adobe languages.

We see in the page content object 2, the familiar "Hello World" string. After the ET operator that ends the Hello World text block we have some new lines of material highlighted. These lines represent the graphics blue star.

Right off we can see why the star is blue since the "rg" operator has arguments specifying 0% red, 0% green and 100% blue.


To look further into how this "graphic" star is drawn we elaborate on the next page.

Page: 34

Author: jameskin

Subject: Note

Date: 4/11/05 5:39 PM

 We already discussed the "graph paper" coordinate system that is the default one used in PostScript and PDF. The origin or zero, zero point is the lower left corner of the drawing area (page) and the units of measure by default are 1/72 of an inch. Both of these things can be changed as we will see in a subsequent example.

In the lower left corner of this page, the graphics operators that draw the blue star are reproduced from example 6. After the operator "rg" that causes the star to be device dependent blue, we see the "m" or "moveto" operator. In PostScript the operator actually is "moveto" but in PDF we try to conserve space so just us "m".

The two arguments to the "m" operator are the distance to move horizontally (X) and the distance to move vertically (Y) from the "current point" which is initially set to (0, 0) at the start of each page.

This moves us to the starting point on the page for drawing our star. Next are four uses of the operator "l" which is the abbreviation for "lineto" meaning imagine a line from the current point to the new point and move the current point, as well. These 4 "l" operators then sketch out a path of four lines giving shape to the star. Notice two things: first so far we have not really drawn anything on the page -- just sketched out an imaginary shape, and second we didn't draw the last or fifth line needed to completely specify the start shape.


The last operator in the graphics group is the "f" operator. That stands for fill. Since the starting point of our imaginary path is not the same as the ending point, the "f" operator adds the fifth imaginary line itself and then it fills in the shape with the current (blue) color.

Page: 35

Author: jameskin

Subject: Note

Date: 4/11/05 5:40 PM


 This page and the next show the complete Blue Star example.

Page: 36

Author: jameskin

Subject: Note

Date: 4/11/05 5:40 PM


 All of the examples shown in this presentation are complete in that all objects are shown. In this case we have already seen the roll that these objects play in example 6 so we just move on to the next example.

Page: 37

Author: jameskin

Subject: Note

Date: 4/11/05 5:40 PM

 OK. Now we have the same star but it is only outlined in cyan instead of being filled with blue.

Just to add a little more complexity to the page we returned to one of the red "Hello World" versions. In fact, this example is based on example 4 which is a device independent red using the L*a*b* color space.


Again, you might want to zoom in for a closer look at the "stroked" not filled star. (Is that something like James Bond's stirred not shaken?)

Page: 38

Author: jameskin

Subject: Note

Date: 4/11/05 5:44 PM

 The device independent red "Hello World" we have seen before in example 4.

Notice something very subtle but important: the color specifications given as part of the star graphics are upper case "CS" and "SC" whereas before in our examples and with the Hello World above they are lower case.

Both PDF and PostScript have the concept of "filling" graphic shapes or "stroking" them. Stroking means to draw the lines but not to fill in the area defined by the lines. Notice that the final operator in this case is "s" which stands for "stroke" instead of the "f" which stands for "fill".

There are two color spaces and two current colors in the PDF graphics state: one for filling and one for stroking. To set the filling ones you use lower case "cs" and "sc" to set the stroking ones you use upper case "CS" and "SC".

Note another important thing. This example has two color spaces used on the same page! One is named /CS1 and the other is named /CS2, being respectively the L*a*b* color space and the calibrated RGB space. We previously used both of these. In addition also note that the rendering intent can be changed whenever necessary to make sure that each object is processed in the most suitable way. Here we direct that the text should be handled as "relative colorimetric" whereas the graphic star should be done using "saturation" intent.

These color space resources are often called color "profiles". This terminology has been promoted by the International Color Consortium or ICC. You may have heard of ICC Profiles. The color space resources in PDF and PostScript serve the same function and are roughly equivalent to the ICC profiles. They aren't literally the same but contain the same or equivalent information. In almost all cases it is easy to convert from a PDF color space resource to an ICC profile and the other way as well.

Most color management systems consider color spaces as being derived from and belonging to color devices. With that view we can consider that the material in this PDF file is associated with two devices one L*a*b* and one an RGB device. Most color management discussions and for that matter color management software, considers a whole job to be in one color space. This is a mistake and a mistake that Adobe PostScript and PDF do not make. Any material in either language can be specified with respect to any color space desired and as many as needed can be used in one document or page.

This is important when assembling material into a document from a variety of sources as is done by Adobe InDesign or Adobe FrameMaker. One would just as soon not convert diverse contributed material to one common color space for the document. Better to leave it be in the space it was created in. PDF and PostScript allow for this. The "CS" and "cs" operators allow one

Comments from page 38 continued on next page


to switch between color spaces defined in resources dictionaries as often as needed. The color space resources can be indirect objects and hence need only appear once in the PDF document and can be referred to by made-up color space names from various resource directories.

One last note. Notice that the color space resources are not very big! Color profiles have a reputation for being large -- it isn't always true.

Author: jameskin

Subject: Note

Date: 4/11/05 5:44 PM


 Again, the remaining objects of example 7 included for completeness.

Page: 40

Author: jameskin

Subject: Note

Date: 4/11/05 5:45 PM

 More Hello World and stars!

This time we have the star both stroked and filled. I would guess this will be easy given what we have already learned.

Notice that the star is filled with yellow and stroked with blue.


Again, you might want to zoom in for a closer look. The neat thing about graphics expressed as coordinates on the page is that no matter how much you zoom in they never get jaggy. One can always draw a smooth line from one point to another no matter what the resolution or zoom factor.

Page: 41

Author: jameskin

Subject: Note

Date: 4/11/05 5:46 PM

 OK. This looks to be based on example 4 with the L*a*b* Hello World.

Here you have to watch the use of upper and lower case letters carefully. The lower case "sc" just after the text block end (ET) is setting the filling of subsequent objects to an L*a*b* yellow. Trust me again (100, 0, 127) is an L*a*b* yellow. In this case the choice of the L*a*b* color space /CS1 that is set within the text block carries out into the graphics portion of the page specifications. There are subtle things like this that require careful reading of the PDF specification.

The "SC" just before the graphics operators is setting the stroke color to a calibrated RGB blue.

The new thing in this example is the final graphics operator "b" standing for "both", where we had previously had an "f" or a "s". This operator directs us to both fill and stroke the star. Since we have two color spaces with a color picked from each this is a well defined operation.


This is another place where PDF and PostScript differ a little. In PostScript there are not two color space and color choices for filling and stroking. There isn't a "both" operator either. The issue is that we want the fill and stroke operators to consume the path so that we can start on our next graphic shape fresh. If we fill the path it will be consumed and not available for subsequently stroking or visa versa. PDF solves this by having two sets of colors one each for filling and stroking and having a "both" operator. PostScript solves it by allowing the path to be saved or duplicated then doing a fill, restoring the path back to life or getting the copy and then stroking it. Just an interesting but small difference.

Note that in this example there are no "ri" or rendering intent operators so that the default "RelativeColorimetric" intent will be in effect.

Author: jameskin

Subject: Note

Date: 4/11/05 5:46 PM


 Again, the last few objects of example 8 for completeness.

Page: 43

Author: jameskin

Subject: Note

Date: 4/11/05 5:48 PM


 The appearance of this page is the same as example 6. Must be showing us an alternative way to do something.

Page: 44

Author: jameskin

Subject: Note

Date: 4/11/05 5:48 PM

 Note that the graphic star is specified with considerably different set of numbers than in example 6.

Here we are showing that you can change the coordinate system to one of your own liking. You don't have to stick with the default one that each page starts with.


We will show this in more detail next.

Page: 45

Author: jameskin

Subject: Note

Date: 4/11/05 5:49 PM

 The light blue lines show the default or original coordinate system and also the star we had drawn in example 6.

The red lines show a new coordinate system that we define with the "cm" or concatenate matrix operator. It takes six arguments. The 4 0 0 4 indicates that distances in the new system will be worth 4 times what they were in both the X and the Y direction (the two 4's) and the zeros indicate that the new coordinate system preserves the rotation and directions of both X and Y (no rotation or skew).

The last two numbers of the six arguments to "cm", namely 315 and 226, give the X and Y coordinates, respectively of the origin (0, 0) of the new coordinate system.


So now we draw the same star using both positive and negative coordinates since the origin is in the middle of the page and we want the star to extend off to the left. Also note that distances specified with this new star are 4 times smaller that previously, since we scaled the system by 4.

Page: 46

Author: jameskin

Subject: Note

Date: 4/11/05 5:50 PM


 This page and the next contain the complete example we just looked at.

Page: 48

Author: jameskin

Subject: Note

Date: 4/11/05 5:50 PM

 Oh my! Now a red heart stroked in black.


I included this to show that curves as well as lines can be used to draw graphic shapes.

Page: 49

Author: jameskin

Subject: Note

Date: 4/11/05 5:52 PM

 One minor but important addition was made to the star. We mention this before we move on to the red heart. Note that a "q" operator has been inserted before the star and a "Q" operator has been inserted after it. These are "save the graphic state" and "restore that save graphic state", respectively. This keeps the coordinate system change in the star specification from affecting our heart specification. The two objects can be drawn independently both assuming the default graphic state settings.


We will go into the drawing of the heart on the next page.

Page: 50

Author: jameskin

Subject: Note

Date: 4/11/05 5:54 PM

 This is the heart graphics formatted a little more clearly. The new operator this time is the "c" which is an abbreviation for "curveto" which specifies a Bezier curve. The name Bezier is normally capitalized because it is the name of a mathematician who promoted this notation.

Bezier curves are supported in both PDF and PostScript and they work the same in both. To get a curve shape involves four points or eight numbers as shown with the blown up first segment of the heart (the upper left section). The current point is the first point. Then the six arguments to the "c" operator provide the other three points as you can see from the diagram. The rightmost argument to the "c" operator is the new current point and the ending point for this section of the shape or path. The two points that are not the endpoints of the shape are called control points and they determine the curvature of the curve. If one considers an imaginary line from a control point to the associated curve endpoint it forms a tangent to the curve at that end point. Another way to think of this is that the control point determines which direction the curve heads away from the end point. The distance of the control point from the end point determines the length of time that the curve continues in that tangent direction before it starts to swing around toward the other endpoint.


If you would like some first hand experience with Bezier curve creation, try Adobe Illustrator because it is one of the fundamental drawing primitives supported by that application. It allows direct manipulation of the control points and you can immediately see the curve change shape.

Page: 51

Author: jameskin

Subject: Note

Date: 4/11/05 5:55 PM


 This page and the next show the complete heart example.

Page: 53

Author: jameskin

Subject: Note

Date: 4/11/05 5:55 PM

 Now we add a sampled image representing the Adobe logo to the page. This page starts with the Hello World and star of example 8.


Notice that this logo image has a black line down the right hand side. We will explain why this is there and we will remove it in a subsequent example.

Page: 54

Author: jameskin

Subject: Note

Date: 4/11/05 5:57 PM

 As you might suspect some more information is added to the page contents that specifies the Adobe logo image. This image specification is directly a part of the page contents object. Later we will see some examples where the image is specified outside of the page content object and just referenced from within the page contents. That is the more normal case: with images being not actually within the page content.

In both PDF and PostScript image data has nothing to do with an output device but is considered as input image sample data. Image data is an array or matrix of numbers representing what color the image is beneath each given spot on the image. The fineness of the grid imagined over the image is the scan resolution for the sampling. Finer resolutions gather more data and more detailed data about the image.

PDF and PostScript assign any sampled image to a shape one unit by one unit (e.g., the image is assumed to be 1/72 of an inch by 1/72 of an inch). One then uses the "cm" operator to convert the page coordinate system to make that image come out the size and shape desired. In this example we scale the coordinate system by 100 in each direction (100 0 0 100) and position it at the point (65, 326). If you look back at the page you will see that the logo is about 100/72 of an inch square and its lower left corner is located at the point (65, 326).

The image sample data is supplied between the "ID" or "image data" operator and the "EI" or "image end" operator. The control information specifying the properties of the sampled data is given after the "BI" or "begin image" operator and before the "ID" operator. The /W or width of the image is 36 samples wide (this is how many characters are on each line including the line return character which is what is causing the black line down the right of the image), /H or height of the image sample data is 32 samples (this is how many lines there are in this sample), /BPC or "bits per component" is 8 or one byte for each sample, and the /CS or "color space" for this image is /DeviceGray or device dependent black and white.


The reason you can actually see the form of the Adobe logo on this page is because the sample values between 0 and 255 do represent printable characters for some values. This is a gray scale image with white or large values in the shape of the "A" and black or lower values filling out the square. I also included line end characters as part of the image data after every 35 sample values in order to cause each scan line of sample to show up here as one line of pseudo text. Those line end characters have a low value and hence show up as black down the right hand side of the image. Notice that the image width is 36 which includes the line end characters. In the next example when we get rid of the line ends, the image sample data width is 35.

Page: 55

Author: jameskin

Subject: Note

Date: 4/11/05 5:58 PM


 Again, the remaining objects for exmple 11.

Page: 56

Author: jameskin

Subject: Note

Date: 4/11/05 5:58 PM


 This is the same example as example 11 but with no black line down the right side of the image.

Page: 57

Author: jameskin

Subject: Note

Date: 4/11/05 5:59 PM

 The difference between this example and the previous one is that the sample data has no line ends within it and hence just wraps at the column width which holds a few more than 35 characters. You can still see the variation in sample values (0 to 255) as they are mapped to printable characters but they don't line up to give the "A" image.

Note that the /W or width parameter is 35.


I wanted to show how the image data really does reflect the image in example 11. If an image is colored or has more than one byte per sample then all hope of pictorially seeing the image in this form as we can in example 11 is really lost because the sample bytes don't have any correspondence to single byte printing characters.

Page: 58

Author: jameskin

Subject: Note

Date: 4/11/05 5:59 PM


 Again, ...

Page: 59

Author: jameskin

Subject: Note

Date: 4/11/05 5:59 PM

 This looks the same as example 12 so it must be to show some other representation of the page.


In fact, we now have a page complex enough where there might be some benefit to compressing the bytes representing the content stream. This is what we show in this example.

Page: 60

Author: jameskin

Subject: Note

Date: 4/11/05 6:01 PM

 The unprintable characters making up most of this page are compressed data for the page contents object.

In example 12 the page contents is a stream containing the "Hello World" string and printing controls for the star and the Adobe logo. All of these things are still in the page content stream here but now compressed using the Flate compression technique. This is the technology used to make PNG files. You will note that the stream dictionary now notes that a /Filter is needed to read the stream and it is the /FlateDecode filter.


If you look back at example 12 you will note that the stream length is 1343 whereas this compressed representation of the stream is only 452 bytes long. That is roughly a 3 to 1 reduction in size. This is an important technique for keeping PDF file sizes small. Unfortunately for the curious it makes reading the files with a text processor like we have been doing not very helpful.

Page: 61

Author: jameskin

Subject: Note

Date: 4/11/05 6:02 PM


 Again,

Page: 62

Author: jameskin

Subject: Note

Date: 4/11/05 6:02 PM

 This page still looks the same as the previous two examples.


This time we are showing some indexing material that is required at the end of all files but which we have not previously shown.

Page: 63

Author: jameskin

Subject: Note

Date: 4/11/05 6:05 PM

 I have been slightly dishonest for pedagogical reasons up to this point.

All of the files I have shown, in fact, technically have been not well formed PDF files. The Acrobat viewers will successfully show all of these examples but only because they are forgiving of imperfect files. In fact, if you try to view one of the example files associated with this presentation the current Acrobat viewers will briefly present a warning message saying that the file has been damaged and is being repaired. They do all repair successfully and display properly.


In the next two slide pages we will show the material that normally is at the end of all valid PDF files. Click on forward, please.

Page: 64

Author: jameskin

Subject: Note

Date: 4/11/05 6:05 PM

 This file has an object 7 which is the "Info" or Information dictionary. This is an optional object and its absence will not generate any warning messages.

It is useful to have this object because it records the last modification date of the file as well as optionally has keys denoting the application that created the file and other such information. Please see the PDF Reference Manual for more details.


The next page shows the new neat stuff.

Page: 65

Author: jameskin

Subject: Note

Date: 4/11/05 6:06 PM

 This is a cross reference table for this example file.

One design requirement for PDF was to allow a viewer to read, say, page 501 in a document of, say, 1000 pages without reading all the file first and without reading pages 1-500 first either. In order to do this there must be some index or cross reference material someplace that indicates which bytes in the file are to be associated with which object. This is the function of the cross reference table shown.

We have already seen a little of this material, namely the "trailer <</Root 6 0 R>>" section. The viewer just does not know where to start without this Root object number. However, given that the objects all have "obj" and "endobj" around them the viewers can read through the whole file and build a cross reference if need be. This is what the file repair amounts to.

The cross reference has a dummy entry for object zero at the start and then seven more, one for each of the objects in the file. So for example the second line is "0000000016 0000 n" which indicates that object one (the position of this line in the list) starts at the 16th byte position in the file. The "00000" is the generation number, being zero for all of the objects in this file. The "n" indicates that this is a valid entry in the table. Note that the first entry has an "f" indicating that it is a dummy non-usable entry.

Other things to note: the Information dictionary (object 7) is referred to by the trailer dictionary. The trailer dictionary also has an /ID key that has an array of two values as its value. These strings represent a unique identification of this file so it can be identified no matter what the file name is changed to. This is important for applications like document data bases and document management systems.

The "startxref" has a number after it indicating where in the file the cross reference table can be found and the "%%EOF" indicates the end of the file.


There is more to all this and the real student will find the details in the PDF Reference Manual.

Page: 66

Author: jameskin

Subject: Note

Date: 4/11/05 6:07 PM


 This example shows that you can place sampled image data into a PDF file only once and then image it onto pages as many times as needed. The Adobe logo is shown twice but the sample image is only in the PDF file once.

Page: 67

Author: jameskin

Subject: Note

Date: 4/11/05 6:08 PM

 The image data is not directly within the page contents object (2) for this example as it is in example 12.

Here the image data is off in its own object (7). The two images on this page are caused to appear by the `/AdobeLogo Do` directives that are in the page contents. As with fonts and color spaces, images not in the page are considered resources. Hence they are given made-up names, in this case, `/AdobeLogo` and the "Do" operator indicates that we should "do" the resource called `/AdobeLogo`.

So as with fonts and color spaces, we look to the resource object associated with this page (object 3) and look in its dictionary for the key `/XObject` which means "external object". We do find one called `/AdobeLogo` and find that it can be found at object 7.


The two images that show up on this page are positioned and scaled by the corresponding "cm" operators and the scaling for the first is prevented from effecting the second by use of the save/restore operators "q" and "Q".

Page: 68

Author: jameskin

Subject: Note

Date: 4/11/05 6:09 PM


 Nothing interesting going on with these old familiar objects.

Page: 69

Author: jameskin

Subject: Note

Date: 4/11/05 6:09 PM

 Here is the /XObject or "external object" that is of /Subtype /Image. There are other kinds of /XObjects. For details look in the PDF Reference Manual.

The image XObject has most of the same parameters that the direct object sampled image had when it was directly imbedded within the page. Attributes like /Width, /Height, /BitsPerComponent and /ColorSpace.


This is the Adobe logo sample image data. It is 1120 or 35x32 bytes in length.

Page: 70

Author: jameskin

Subject: Note

Date: 4/11/05 6:10 PM

 This example introduces more than one page into our document for the first time.

We changed our old friend "Hello World" to "Page 1" and "Page 2" to distinguish the two otherwise identical pages.


Another feature of this document is that the Adobe logo appears four times, twice on each page, but the sample data for it are in the PDF file only once. This shows that Xobjects can be referenced globally within the document and can be used on any page.

Page: 72

Author: jameskin

Subject: Note

Date: 4/11/05 6:10 PM

 Here we note that the /Pages object 5 has an array of /Kids with two entries. The first for the first page pointing to object 2 and the second for the second page pointing to object 8. Note two that the /Count key in this Pages object is 2 indicating that there are two pages in the document.

Otherwise, pages 1 and 2 represented by objects 2 and 8 are simple copies of the page we saw in example 14.


Note that page 1 has a resource object number 3.

Page: 73

Author: jameskin

Subject: Note

Date: 4/11/05 6:10 PM


 Nothing new here.

Page: 74

Author: jameskin

Subject: Note

Date: 4/11/05 6:11 PM

 This object 8 is page 2 of the document. It has an associated resource object 9 and page contents object 10. These are very similar to objects 2 and 3 for page 1.


In fact the both resource objects (9 and 3) point off to the same font object (4) and the same indirect Xobject (7).

Page: 75

Author: jameskin

Subject: Note

Date: 4/11/05 6:11 PM


 Here is the shared font object 4 that is referred to by both pages' resource objects.

Page: 76

Author: jameskin

Subject: Note

Date: 4/11/05 6:11 PM


 And here is the sampled image data object 7 that is also referred to by both pages' resource objects. This image data is used four times in this document. Twice on each of two pages.

Page: 77

Author: jking

Subject: Note

Date: 4/25/05 4:36 PM

 Now we are going to switch gears a little and talk about using PDF files to archive documents.


There has been a subset of PDF defined called PDF/A. If one sticks to this set of features in a PDF file it will stand a much greater chance of being useful as an archived document.

Page: 78

Author: jking

Subject: Note

Date: 4/30/05 11:37 AM

 These features listed can cause issues when using PDF as an archive format. For example device dependent color means that the colors could change because their specification is linked to the particular colorants in the device it is being printed or display upon. An archived document should be presented the same in all situations. So it should not use device colors. Similarly if the same fonts the author used are not used to display or print the document the results are unpredictable. So if one embeds all the fonts within the PDF file the results will always look the same.

So far, we have not standardized on PDF's own way to do sound and movies. So what is used are formats specific to particular operating systems. If you open a file designed for one system on a different system you may not be able to hear the sound or see the video.

JavaScripts can change the appearance of the displayed material dramatically and conditionally. If a PDF contains JavaScripts it will not always provide a reliable archive document.

On the other hand, there are some features that are favorable to archiving documents. If the PDF file has metadata associated with it and in standard ways, it will be much more useful in the future. And metadata can help in indexing and searching groups of documents.

Structural information can be included in PDF file giving semantic information about what you see on a page. For example tagging can mark heading, headers and footer, page numbers, etc. This make the information within the PDF file more re-usable.


If one wants to archive forms documents it is best if the places where one can type into the document, the form fields, locked or "frozen".

Page: 79

Author: jking

Subject: Note

Date: 4/30/05 6:31 AM


 This page is rather self describing. It gives some URLs where you can get more, and more up to date, information on the PDF/A standard.

Page: 82

Author: jking

Subject: Note

Date: 4/30/05 11:37 AM

 Drag your mouse over the "Name" form field. It should change to have a yellow background. But also notice what happens to the "Refund value" form field as you do.

If one were to archive such a file it may be hard to determine exactly what the refund value was supposed to have been.


Unfortunately if you are archiving files, or if you are doing forms work, fortunately, these kinds of PDF file are quite easy to create.

Page: 83

Author: jking

Subject: Note

Date: 4/25/05 6:12 PM

 We will show you how JavaScripts are stored in a PDF file but we will also learn the basics of how forms are defined in PDF.

This page starts off much like the "Hello World" page but it has the addition of the Forms field definitions. You will note that the Page dictionary (object 1 0) has a new key, /Annots, which points off to two annotations on that page that represent the areas where typing can take place, the form fields. Form fields are implemented as a special case of annotations.

Look at object 5 0 for example. Besides defining the area into which the user can type a "FullName" value, it refers to two JavaScripts one is object 10 0 as the value of key /E and the other is object 11 0 the value of key /X. Those keys are for mouse enter and mouse leave the field area, respectively. You see that they change the color of the "FullName" field and change the value of the "Worth" field as we experienced when mousing over the page.


Object 5 0 also has an /AP key to hold a default appearance of what that field should look like and it is given as object 9 0 on the next page.

Page: 84

Author: jking

Subject: Note


Date: 4/25/05 5:54 PM

 Objects 12 0 and 13 0 mimic the objects 5 0 and 9 0 and define the "Worth" field.

Author: jking

Subject: Note


Date: 4/25/05 5:53 PM

 Object 9 0 contains a fragement of page contents that fills and strokes a rectangle to show the form field.

Author: jking

Subject: Note

Date: 4/25/05 6:07 PM


 Besides each page referring to the form fields that it contains with the /Annots key, all of the form fields can be located by following the /AcroForm key in the Catalog dictionary (on the next page and it refers to this object 7 0).

Page: 85

Author: jking

Subject: Note

Date: 4/30/05 11:37 AM


 This is all normal stuff except for the /AcroForm key in the Catalog dictionary.

Page: 86

Author: jking

Subject: Note

Date: 4/30/05 11:36 AM


 One can add structural information to a PDF file to provide semantic descriptions of what is displayed on each page. We will go over this in the next few slides.

One particular way to structure a file is called a "Tagged PDF". This represents particular element set that has been defined by Adobe. Both the means to add structural information to a file and the particular tag set that makes a PDF file a "Tagged PDF" are spelled out in great detail in the PDF Reference.

Author: jameskin

Subject: Note

Date: 4/30/05 11:36 AM


 This is a very simple file that I made in Microsoft Word. It has two levels of text content and we will show how the headings for these two levels are "marked up" using PDF structuring mechanism.

Page: 88

Author: jameskin

Subject: Note

Date: 4/30/05 6:44 AM

 The structure information is based on the full document not on individual pages. For that reason the root of the structural information is found in the /Catalog dictionary and uses the key /StructTreeRoot to point off to the top level dictionary of the structure information at object 18 0.

Object 18 0 points to a /ClassMap (22 0) and a /RoleMap (19 0).

The Role Map provides an element naming alias dictionary to relate the element names that the application wants to use with the Adobe defined names. For example, in the Word file we had called the highest level heading a "Heading 1" and Adobe defined the highest level heading as an "H1". So the association between these equivalent notations is given in this dictionary. (Note that the blank character between "Heading" and "1" is prerepresented by its hexadecimal value "#20".


The Class Map (22 0) has 4 entries for the four types of elements found in this file: Heading 1, Heading 2, Heading 3 and Normal formatting. The formatting rules for each of these is pointed to by keys in the Class Map dictionary (23, 29, 34, and 26 respectively). These are shown on the next page.

Page: 89

Author: jameskin

Subject: Note

Date: 4/30/05 11:36 AM

 Each of these four dictionaries define the formatting directives used by Word to layout the Heading 1, 2, 3, and Normal element types.

These settings are not used to display this page's content but can be used by other application that may want to re-layout the content. These directives are also used by Acrobat to perform a reflow operation for better reading ease if the user directs it to do that. You can try it using View-> Reflow.


I will not go into to details of exactly what "SpaceAfter" etc. mean, but you can find that in the PDF Reference. What is shown here is not the complete list.

Page: 90

Author: jameskin

Subject: Note

Date: 4/30/05 8:34 AM

 Object 55 0 is the page content stream for this single page. You can verify that by starting with the /Catalog, finding that 10 0 is the /Pages dictionary (10 0) which references the single /Page dictionary (1 0) which references this page contents 55 0. You will also notice that it contains the text strings that are displayed on the page, e.g., "This is a paragraph".

You will note that there are many /MCID keys shown within the page content that I have highlighted in red. The overall construct of which these keys are a part is delineated by "marked content" operators. These are page content operators just like the lineto and moveto operators we saw earlier except that these marked content operators cause no imaging on the page. They are present merely to delineate a section of operators that do draw on the page and when a viewer is displaying the page these operators are simply ignored.

The first marked content operator is the "BDC" which is a begin marked content operator and that is matched in each case by a "EMC" or end content operator. Between those two operators is the content being marked. The BDC operator has two operands that precede it. The first is the "tag" which is an arbitrary PDF name and plays a role much like an element name would in XML. All of these marked sections are marked as /P's. The second operand is a dictionary and in our case it contains a value for the marked content ID or /MCID key. Each of our marked sections is given a unique numeric ID starting at 0 and going up to 7. There are 7 marked content sections in this document.


These are marked so that our structural information may be associated with them denoting them as Heading 1, etc.

Page: 91

Author: jameskin

Subject: Note

Date: 4/30/05 8:55 AM

 You will note a set of dictionaries on this page and the next in which I have highlighted a /K key (i.e., 20 0, 24 0, 27 0, 32 0, etc.). These dictionaries are what are called structure elements and make up the structure tree. This simple one level 7 way branching tree is created by the following links: Object 41 0 has a /K key whose value is an array listing all of the structure elements forming the 7-way branch. The /K key (41 0) found in the StructureTreeRoot dictionary (18 0) referenced from the Catalog (16 0).


Within each of these structure element dictionaries the /K keys have the same values zero through 7 as the /MCID keys and serve to link the marked content sections identified on this page to the kind of formatting element that they were derived from. For example the marked content section 1 (/MCID 1) on the preceding slide causes the output "This is a paragraph." to be imaged on the page. By finding the associated dictionary with the "/K 1" we see that this line of text is in the "Normal" class, the value of the /C key in that same dictionary.

Page: 93

Author: jameskin

Subject: Note

Date: 4/30/05 9:18 AM

 Object 41 0 is the root of the structure tree which in this case is a simple one level 8-way branching tree. The eight sub elements are referenced by the array elements in the /K key.

Object 43 0 is a "number tree" which is a general construct used within PDF to do numeric lookups efficiently. The /Nums key indicates that the numbers that can be looked up start at "0" and are found in order in object 44 0. So the 0-th object in this number tree is 20 0, the 1st is 24 0, the 2nd 27 0, etc.

So if you are scanning the page content and want to reach the structure element for the content marked as /MCID 1 one can use this number tree to look up the entry for 1 to find the dictionary 24 0 which indicates that this content is of type "Normal".

The number tree can be reached from the StructureTreeRoot using its /ParentTree key value.

The remaining objects on this slide should be familiar to us by now as the /Pages, /Page and font dictionaries.

We have only shown one simple example of how a structure tree is associated with the content of a document. If this document had more than one page we would still only have one structure tree and the marked content on the various pages would be linked to it using the marked content unique IDs, MCIDs just as in this example.

The PDF Reference has a well written description of these structuring mechanisms with further examples.


This sample document is actually also a "Tagged PDF" because it follows the requirements of having a structure tree which uses standard element definitions, having the text be laid out in the page content in normal reading order, and obeying some other rules for being able to decipher the glyph strings into Unicode properly.

Page: 95

Author: jameskin

Subject: Note

Date: 4/30/05 11:36 AM

 We will use the same one page text document we used in the previous example to show how document metadata is included into a PDF file using the Adobe Standard "Extensible Metadata Platform" or XMP.

XMP is used across Adobe's products for representing metadata in an XML markup language. That language is based upon the Dublin Core vocabulary and uses RDF for simple relationships. A large set of basic vocabulary items are standardized in XMP but it also allows for further items to be defined by use.


In order to facilitate finding and reading the XMP within some more complicated file format, Adobe has defined an XMP "packet" which is recorded in clear text (i.e., uncompressed) and which always begins with the string "W5MOMpCehiHzreSzNTczkc9d". In this way a simple program can find the metadata packets within a PDF file, a Photoshop file, an InDesign file, etc. without being required to know how to read and parse the full contents of those different kinds of files. The program just scans through the bytes of the file looking for this unusual and hopefully unique string of bytes. It can then isolate the XMP packet and read it. It is also possible to update the packet as long as one does not change its length and keeps it in exactly the same position within the containing file.

Page: 96

Author: jameskin

Subject: Note

Date: 4/30/05 11:35 AM

 The document level metadata is identified in the document Catalog by the key /Metadata and in this case it refers to object 56 0. We see that that object is a stream that is 3931 bytes long and the bytes of the stream are an XML file.

Complete documentation for XMP as well as access to a developers Software Developer's Kit (SDK) can be found on Adobe's website.

Suffice it to say here that this XML markup language uses simple RDF notation (<http://www.w3.org/RDF>) for relationships and some basic Dublin Core vocabulary (<http://dublincore.org>).


As we noted earlier this XMP packet or stream object can be located in one of two ways. The first way is using the knowledge we have of PDF and finding the /Metadata key in the /Catalog dictionary. The second way is simply to scan the whole file for the string "W5M0MpCehiHzreSzNTczkc9d" which identifies all XMP packets.

Page: 97

Author: jameskin

Subject: Note

Date: 4/30/05 11:52 AM

 This shows the second half of the XMP packet ending in the normal "endobj" string after the 3931 bytes of the packet. Then we have the remaining parts of the file as we examined in the previous section on Structure in PDF.

If you jump ahead 7 slides, I have laid out the XMP packet in one column which is much easier to examine. So click forward 7 slides.


<http://www.adobe.com/products/xmp/>

Page: 104

Author: jameskin

Subject: Note

Date: 4/30/05 11:53 AM

 Apparently this document has been assigned the unique ID (UUID) of "ad0dab0d-84e1-11d8-8dfc-0030677a5e6" as we can see from the RDF "about" attributes on most of the Description elements.

You can see that the Producer of this file was Acrobat Distiller 6.0.1 running on Windows, and that the company where it was produced was Adobe Systems. These are specified using the "pdf" and "pdfx" namespaces respectively, meaning that those sections of the vocabulary hold. It also uses the "pdfx" SourceModified element.


I would think that the use of the photoshop namespace would not be necessary for this document and we could drop the elements at the bottom of this page. (I'll have to check up on that.)

Page: 105

Author: jameskin

Subject: Note

Date: 4/30/05 11:53 AM

 Here you can see that the namespace "xap" is used to define the CreatorTool, ModifyDate, CreateDate, and MetadataDate. Note that these dates use a different format representation than the SourceModified data on the previous slide which uses the pdfx namespace.

XAP is an Adobe defined namespace that uses a vocabulary defined by Adobe.


The next section uses the namespace "xapMM" which stands for Media Management. This is another vocabulary that Adobe has defined and here we see it is specifying that we are using version 4.

Page: 106

Author: jameskin

Subject: Note

Date: 4/30/05 9:49 AM


 Here we see the use of the Dublin Core namespace (dc) to define a title, and creator.

Page: 107

Author: jameskin

Subject: Note

Date: 4/30/05 10:09 AM

 If you actually worked your way to this slide through all the preceding slides you have my admiration.

The PDF file specification and the Adobe Acrobat product line were introduced together for the first time in June 1993. Since then the PDF Reference specification has been revised 5 times now at PDF 1.6 and the Acrobat product line has been enhanced 6 times now at Acrobat 7.

Any valid PDF file written at anytime in the past will be displayed properly using Acrobat 7. There are some PDF 1.6 files that will not display properly if one were to still have and use an Acrobat 1.0 viewer. I hope no one is that far behind. Another way to say this is that PDF 1.4 is a subset of PDF 1.5 which is a subset of PDF 1.6, etc. We do not change existing definitions and we have almost never removed any features. when a new version is created. (We actually have removed a few minor things that were never used but I won't tell you what there were.)

In our sample files displaying text, graphics, and images, we saw how carefully the PDF language has been defined and how precisely material can be specified for output. We call this high fidelity. You also can see that nearly all the specifications within a PDF file are independent of the exact way the output will be produced, be it a color display, desktop printer or very high resolution imagesetter.

The definition of PDF does not depend upon any assumptions about the operating system on which the files will be read or written with the exception that the files are assumed to be stored on a random access device where individual bytes from the file can be read in any order. Adobe is trying to supply Acrobat software on all those platforms where it is needed and you will see us try even harder in the immediate future.

Although not perfect, the PDF Reference manual is about as precise and complete as we know how to make it. So anyone can write applications for writing, reading and processing PDF files and all of those applications will inter-operate with each other.

Thanks for your interest,
Jim King -- 4/30/2005
