

Notes from jcking00.pdf

Page 1

Note 1; Label: Jim King -- Adobe Systems; Date: 8/6/98 3:13:45 PM

Thanks for looking at this presentation.

Nearly each page/slide has an annotation of the kind of things I would say when giving this presentation in person.

The best way to review this slides may be to print out the annotations and read them as you use the full screen area to view the slides. If you have Acrobat Exchange you can make a file of just the annotations by using Tools->Summarize Notes from the menu items. Then you can print the notes.

I hope you enjoy this material. I had a pretty good time putting it all together.

Thanks,
Jim King 8/6/98

Page 2

Note 1; Label: Jim King -- Adobe Systems; Date: 8/6/98 3:14:38 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 Acrobat Reader or Acrobat Exchange.

You can also look inside the files yourself by using any text editor like Notepad, Simpletext, 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 -- 8/6/98

Page 3

Note 1; Label: Jim King -- Adobe Systems; Date: 8/6/98 3:16:14 PM

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. That version of PDF is 1.0

Today we are up to version 1.2 and it can be found on Adobe's World Wide Web site as noted on the slide.

This presentation together with the example PDF files will be on the ICPS Web site as shown.

Page 4

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:02:01 PM

This is a very simple PDF page (example 01). It has the phrase "Hello World" in black in 24 point Helvetica. Note that it is placed on an 8.5 by 11 inch page.

Page 5

Note 1; Label: Jim King -- Adobe Systems; Date: 3/12/98 2:48:06 PM

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 has 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.

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 6

Note 1; Label: Jim King -- Adobe Systems; Date: 3/12/98 2:49:14 PM

Note that the other 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. This is the current specification and is the version supported by the Acrobat 3.0 and 3.01 products.

Page 7

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:06:32 PM

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 proceeding 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 documents "/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 8

Note 1; Label: Jim King -- Adobe Systems; Date: 3/12/98 2:53:08 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 9

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:07:52 PM

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 10

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:01:04 PM

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 11

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:09:35 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 old 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 12

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:10:21 PM

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 600. 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 13

Note 1; Label: Jim King -- Adobe Systems; Date: 3/12/98 3:02:36 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 14

Note 1; Label: Jim King -- Adobe Systems; Date: 3/12/98 3:03:56 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 our sample 8.5 by 11 inch page there are 612 units horizontally and 792 units vertically. Think of graph paper.

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 (600). So our string is to be placed 260/612 -ths in from the left and 600/792 -ths up from the bottom.

Page 15

Note 1; Label: Jim King -- Adobe Systems; Date: 3/12/98 3:06:46 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 16

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:15:01 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 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 17

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:16:09 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 18

Note 1; Label: Jim King -- Adobe Systems; Date: 3/12/98 4:43:31 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 19

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:19:22 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 20

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:22:19 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 21

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 9:10:48 AM

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

Page 22

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:24:55 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 23

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 9:15:48 AM

Another red "Hello World"?

Page 24

Note 1; Label: Jim King -- Adobe Systems; Date: 8/6/98 3:52:56 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 made up name /CS1. This was a name of my choosing. It could just as easily been /mycolorspace or /X or whatever.

If you click on this color space definition the Acrobat viewer will jump to a page with that formatted out into a more readable form.

To return to this page, click on the left side of that page.

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.

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 25

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 9:35:55 AM

A third red "Hello World"?

Page 26

Note 1; Label: Jim King -- Adobe Systems; Date: 8/6/98 3:56:06 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 click on that part of the page. To return to here click on the left side of that 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.

Page 27

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 9:49:12 AM

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 28

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:32:11 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, click on that area of the page. To return to this page, just click anywhere on that subsequent page.

Page 29

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:39:15 AM

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 30

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:42:54 AM

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 31

Note 1; Label: Jim King -- Adobe Systems; Date: 8/6/98 4:06:35 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 colormetric" whereas the graphic star should be done using "saturation" intent.

If you want to see the better formatted versions of these color space specifications just click on them. To return here click on the right side of those subsequent pages.

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 on a RGB device. Most color management discussion and for that matter color management software, consider 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 PageMaker 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 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.

Page 32

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:01:35 AM

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

Page 33

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:04:20 AM

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 34

Note 1; Label: Jim King -- Adobe Systems; Date: 8/6/98 4:11:48 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 "RelativeColorMetric" intent will be in effect.

Page 35

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:17:47 AM

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

Page 36

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:14:03 PM

The next two pages show a PostScript file that images the same page contents as the example 8 PDF file. It is interesting to see the variations in the two representations of the same page.

Notice the lack of objects in the PostScript.

Page 38

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:18:46 AM

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

Page 39

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:42:22 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.

Click on the star graphics specification to link to a page where this is explain with a diagram.

To return here, just click anywhere on that page.

Page 40

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:28:53 AM

Again, the remaining objects for example 9.

Page 41

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:46:24 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 42

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:47:43 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.

To go to an explanation of the heart click on its specification in the third column.

To return here, click on the two pages you will visit.

Page 43

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:57:56 AM

Again, the remaining object for example 10 for completeness. We have seen all of these before in previous examples.

Page 44

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:48:34 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 45

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:52:22 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, 672). 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, 672).

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 46

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 12:18:10 PM

Again, the remaining objects for exmple 11.

Page 47

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 12:18:54 PM

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

Page 48

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:54:03 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 49

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 12:24:13 PM

Again, ...

Page 50

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 12:26:35 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 51

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 10:55:44 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 LZW compression technique. You will note that the stream dictionary now

notes that a /Filter is needed to read the stream and it is the /LZWDecode 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 52

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 2:06:04 PM

Again,

Page 53

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 2:07:05 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 54

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:15:42 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 55

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 2:15:46 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 56

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:18:47 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 an 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 57

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:19:21 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 58

Note 1; Label: Jim King -- Adobe Systems; Date: 8/6/98 4:19:13 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 59

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 3:01:58 PM

Nothing interesting going on with these old familiar objects.

Page 60

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:22:07 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 61

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 3:08:36 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 63

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:23:30 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 64

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 3:11:43 PM

Nothing new here.

Page 65

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:24:14 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 66

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 3:14:14 PM

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

Page 67

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 3:15:19 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 68

Note 1; Label: Joe Doe - Adobe; Date: 10/20/97 11:59:37 PM

A second annotation.

Note 2; Label: Jim King - Adobe; Date: 10/20/97 11:59:37 PM

An annotation on the Hello World page.

Page 69

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:26:33 PM

I put no presentation text annotation on the preceding page because it was very confusing since it is an example of pages with annotations on them.

That page has two text annotations. Here is the way that information is represented. Since annotations are associated with a particular page they are "hung off" of the /Page object (1) for the page they are on using a new key in the /Page object dictionary /Annots.

The /Annots key has as its value an array which in this case has two entries, a reference to text annotation object 7 and 8. Note that these /Annot objects have as their /Subtype /Text. We will see a /Link annotation next and there are several other kinds. These are text annotations used to place text notes onto the page.

If you compare the /Annot object 7 and 8 with the annotations shown on the previous page, it is pretty obvious that these objects contain the information needed to display the annotations. There is a key /Open indicating whether the note is to be shown opened or closed when the page is shown. There is positioning information for each via the /Rect key, the color of the note is indicated by the /C key and is in the device dependent RGB color space. The title string (Jim King -- Adobe) is found in the /T key, etc.

Please see the PDF Reference Manual for more details.

Page 70

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 3:25:18 PM

Nothing interesting here.

Page 71

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 3:27:29 PM

This page is example 1 with a link annotation place onto it. This link annotation is boxed in a green outline and when you click within the box the page pan/zoom of the viewer is adjusted to show the page in "fit the width" mode.

Try clicking on within the green box that outlines "Hello World". You can return to normal viewing by using the View->Fit Page menu item in the Acrobat viewer.

Page 72

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:29:17 PM

This annotation also "hangs off" the /Page object 1 in this example. Note that the annotation object is of /Subtype /Link.

It has a border key indicating that the border should be painted in green. It also has a key /InvisibleRect /I indicating that the rectangle should be displayed. Other setting will make the

rectangle invisible.

The other notable thing about this annotation object 7 is that it has an /A key which stands for "action" and that key references an "action" object number 10.

This object 10 gives the directives as to what the viewer should do when this action is taken (it is taken when one clicks on the annotation). It is a /GoTo action and the /D or destination is the page whose /Page object is object 1 and to display this page using the /FitH which is an abbreviation for "fit horizontally" and the number 650 indicates where the top of the window should be placed in the vertical direction.

If this document was more than one page the page object reference could reference any page in the document. Other forms of actions can cause links to other documents and many other functions. Please see the PDF Reference Manual for the large variety of things annotations will support. Each one has different keys that determine the parameters and many have various subordinate objects. This is one of the areas of the PDF specification which is stuffed with fine details which provide a lot of power but also make the manual fatter.

Page 73

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 3:37:16 PM
Nothing new.

Page 74

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:30:17 PM

This is a three page document. Cause your Acrobat viewer to show the table of contents or bookmarks by clicking on the second icon from the left in the tool bar. (Well it should be there if the Acrobat 3.0 defaults are in place.)

This will display some bookmarks in a small vertical window on the left. Notice that there are seven entries for this three page document. There are the strings like (BookM 1), (Top of Page 1), etc. that represent to the topics to which the bookmarks or table of content refer.

If you click on the bookmarks there will be a /GoTo action executed to take you to the proper page and view. Try it.

Page 77

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:31:48 PM

Book Marks or Table of Contents are a document wide data structure so they are "hung off" of the /Catalog object. This is the object that the /Root points to as the document starting place. Here the data structure representing the bookmarks are pointed to by the /Outlines key and in this case point off to object 14. Note that the terms "table of contents", "bookmarks" and "outlines" all refer to this one same mechanism.

The other new key in the /Catalog object the /PageMode key indicates that this document should be opened into the outline showing mode when it is opened. If you were to take the example 18 PDF file by itself and open it in an Acrobat viewer it would open with the table of contents exposed because of the setting of this key to /UseOutlines.

Note that the /Kids key of the /Pages dictionary now shows pointers to three pages (objects 1, 8 and 11) and the /Count key is 3.

Page 79

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 4:02:39 PM

This is the table of contents structure starting at object 14. Click anywhere on this page and you will jump to a page showing the relationship among all of the various objects making up the table of contents or outlines structure. Click on that page to come back here.

Note that the strings that show up in the table of contents window are held in these objects one string per one entry per one object.

Page 80

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 4:08:30 PM

More of the objects making up the outlines structure.

Page 81

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 4:08:41 PM

And more.

Page 82

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:33:40 PM

This is a three page document demonstrating how the thumbnails feature of PDF is represented within the PDF file.

Click on the tool bar (third from the left) to open the thumbnail viewing window. This shows that these three pages have thumbnails that have been created for them. They are small images that look like the page in miniature. You can navigate around the page and the document by clicking on the thumbnails in the thumbnail viewing window. If you have never done this try it.

Page 85

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:34:38 PM

Since the thumbnails are associated with specific pages, the thumbnails themselves "hang off" of the individual /Page dictionaries by means of the key /Thumb. See for example where the page object 1 refers to its thumbnails as object 32.

Before we leave this page, look at the /Catalog object number 6. It has a key /PageMode which indicates to /UseThumbs which means that when this document is opened the thumbnail subwindow should also be shown. Note that the /PageMode key is the same one that had as its value /UseOutlines in example 18. You cannot both show the thumbnails and the outlines at the same time.

We will move on to object 32 and see what a thumbnail object looks like.

Page 86

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 4:16:31 PM

Just the same old pages.

Page 87

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:36:38 PM

Here is object 32 in the lower right corner. Note that it is a stream object that has been run through two filters. Thumbnails are small image objects. They have most of the same parameters that image streams do.

To decode this thumbnail object 32 stream, one has to run the stream through the /ASCII85Decode filter and then through the /LZWDecode filter. The LZW compression makes the thumbnail image smaller and the ASCII85 converts it so that all the characters are printable and might not get

damaged in transit so easily by 7-bit ASCII processing steps.

These thumbnails exhibit one other rather interesting design feature of PDF. One requirement that was placed on the designers was that the PDF files should be easy to create in one pass from front to back. This requirement, in conjunction with the requirement to compress large streams causes a problem. For many compression methods one does not know how big the resulting compressed stream will end up until the data has been compressed. This means that if the /Length of the stream has to be put out in the stream dictionary that occurs before the stream data one would have to buffer up the compressed data before writing it out in order to find out how long it is. An easy solution is to allocate a new object to hold the length value that will be put out after the stream data.

You will see that the stream dictionaries indicates another object where the length can be found. For example, thumbnail object 30 on this page refers to /Length object 31 which contains the length of stream 30 (544) and 31 occurs after 30 in the file.

The program reading this file does so as follows: read the dictionary of object 30 and determine that the length of the stream we haven't yet read can be found in object 31. Don't start reading the stream data just yet, but instead read object 31 and get the value 544. Now go back to object 30 and read its stream data knowing exactly how long it is and how much to read. Remember the PDF file format was designed to be read from a random access device which allows one to read any span of bytes in the file at any time.

Page 88

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 4:31:44 PM
More thumbnail images.

Page 89

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 4:31:54 PM
And more.

Page 90

Note 1; Label: Jim King -- Adobe Systems; Date: 8/6/98 4:24:58 PM

Our last example is showing a glimpse of the Acrobat forms feature. Here we have a form asking for us to type in our Name.

The heading "Name:" and the underlines following it are part of a normal page and referred to as the page's "artwork". That part of the page can be made with any page creation tool.

After the resulting page has been made into a PDF file, that file is opened in Acrobat Exchange and a form field can be defined where a subsequent person reading the document can fill in their name. The tool in Exchange for creating the to-be-filled-in field can specify the properties the field should have like colored background as in this case a purplish color and an outline color, in this cases blue.

The Acrobat 3.5 Forms feature has many exciting new form features. We aren't even scratching the surface with this example but it does show how information like form defining data can be structured and contained a PDF file. We also use this example to show what Form Data Format or FDF data looks like.

Page 91

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:40:32 PM

The form field in this example is yet another example of a page annotation. This time the annotation is of /SubType /Widget. As with other annotations such as text and link that we saw earlier, the position on the page and other pertinent information is stored in keys in the annot dictionary object.

The field in this case has been named the "Fullname" field and that is under the /T key. The /AP key which is an abbreviation for "appearance" refers off to object 14.

That object is a stream object whose dictionary defines many more things pertaining to the appearance of the form field on the page. Such as the box size where the typing is allow (/BBox) and the color that the typed in material should be display with. The stream itself contains page content operators indicating what font and color the typed in text should be and creating and filling and stroking the box ("re" is an operator that creates a rectangle path) that contains the typed in material. Lots and lots of details. Of course all carefully documented in the PDF Reference Manual.

Page 92

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 4:47:33 PM

More details about this one field and how it is allowed to be filled in.

Page 93

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 4:48:11 PM

This is continues the previous example with the field having been filled in with my name, Jim King.

Page 94

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 4:49:45 PM

The main thing to notice here is that the /Annot object /Widget has a key holding the value (/V) of what was typed in, namely the string (Jim King). Most of the other material remains as before the fill-in was done.

Page 95

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 4:51:33 PM

Notice that the page content fragment now also includes the string that was typed in. Note that it is within a text block bracketed by "BT" and "ET" just like our original Hello World example was.

Page 96

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 4:51:51 PM

Standard stuff.

Page 97

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:44:59 PM

This is a text display of an FDF or Forms Data File. This is not really a PDF file -- it is a cousin, an FDF file.

Fields or places where a person can enter data into a form have "field names". In our example we had one field and we had named it "FullName". The FDF file can be created from a filled in form when using Acrobat Exchange from the menu item "File->Export-> Form Data". This takes each field name in the document and its associated filled in data and writes them as one entry in the FDF file.

This allow processing programs to deal only with the values that were typed in and not the artwork that makes up the appearance of the form but is of no interest or use to a processing program. One can export the data and send it to a processing center or to another user that has a form with the same fields defined who can import the FDF data to fill out the form. The FDF files will generally be much much smaller that the filled out PDF file which includes the artwork and the field specification data.

Notice in this example the substantial information in the file is that there is a field named (/T key) (FullName) and it has had a value (/V key) with its value the string (Jim King) which is what was

typed in.

It also refers to the original form PDF file with the /F key. This allows one to click on the FDF file and the Acrobat viewer will attempt to find the file the data came from, open it and import the FDF data into it.

Page 98

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:47:58 PM

We have spent quite a long time going through examples of PDF files. If you followed along through most of the examples you will by now have a pretty good feel for the rules and constructs that are used when building a valid PDF file. It is a powerful notation easily expressing values of numbers, strings, and names. These can be combined into composite elements such as dictionaries and arrays. These in turn provide the contents of PDF objects.

The objects can refer or point to one another to form complex relationships and data structures. Streams of arbitrary unstructured data can be defined. The cross reference table that occurs at the end of the PDF file allows program to read selected objects out of the file at arbitrary positions. The objects don't have to be in any particular order or place within the file. The cross reference table determines where each object has ended up in the file.

This notation is powerful enough to represent many other kinds of structured data beside a PDF document. In fact Adobe has begun to use the notation for at least two other purposes: one for FDF files to hold the field name, field value pairs that result from filling in an Acrobat Form, and the second which we haven't talked about in this presentation: the PJTF or Portable Job Ticket Format used to control the workflow and post-processing steps applied to PDF jobs.

PDF files require that they have a /Root which points to a /Catalog which points to a /Pages object which points to the documents pages. Not all uses of this notation should have to follow those document rules. In fact, both the FDF and the PJTF do not. So to be perfectly logical, each of PDF, FDF and PJTF are using a notation and specializing it for their particular needs. For this reason I am trying to coin the term SDF or Structured Data Format for the general use of this notation and reserve PDF for the specialization having to do with representing documents in this particular way. Thus this diagram!

Page 99

Note 1; Label: Jim King -- Adobe Systems; Date: 8/6/98 3:16:14 PM

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. That version of PDF is 1.0

Today we are up to version 1.2 and it can be found on Adobe's World Wide Web site as noted on the slide.

This presentation together with the example PDF files will be on the ICPS Web site as shown.

Page 101

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:49:08 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.

To return to example 03 click on the left side of this page. To return to example 07 click on the right side of this page.

Page 102

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:50:07 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.

To return to example 05 click on the left side of the page. To return to example 07 click on the right side.

Page 103

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:51:08 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.

Click on the right side of the page to go to more explanation about Bezier curves. Click on the far left of the page to go back to example 10.

Page 104

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:53:53 PM

Click anywhere on this page to return to example 6.

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.

Click anywhere on this page to return to example 6.

Page 105

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:55:24 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 550, 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 than previously, since we scaled the system by 4.

Click anywhere on the page to return to example 9.

Page 106

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:57:14 PM

Click on the right side of this page to go back to example 10. Click on the left side of the page to go to the clearer formatting of the heart graphics directives.

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.

To return to example 10 click on the right of the page.

Page 107

Note 1; Label: Jim King -- Adobe Systems; Date: 3/13/98 11:58:11 PM

Click anywhere on this page to return to example 18.

This diagrammatically shows how the objects 14-29 reference each other in order to make the logical structure and nesting of the outlines as shown in the table of contents window.

The keys /Next, /Previous, /Parent and /Last are used to form the outline structure. The /A key is used to refer to the action that will take the viewer to the appropriate page and view within the document.

If you really want to get into this you can go back and forth between this diagram and example 18 and see how these 15 objects form the table of contents structure.

Click anywhere on this page to return to example 18.